TEK REFERENCE MANUAL

Part No. 070-4729-00
Product Group 61

8560 Series Multi-User Software Development Unit

TNIX
8560 Series System Reference Manual

TNIX Version 2

Please check for change information at the rear of this manual

First Printing SEP 1983

Tektronix
COMMITTED TO EXCELLENCE

LIMITED RIGHTS LEGEND

Software License No.

Contractor: Tektronix, Inc.
Explanation of Limited Rights Data Identification Method
Used: Entire document subject to limited rights.

Those portions of this technical data indicated as limited rights data shall not, without the written permission of the above Tektronix, be either (a) used, released or disclosed in whole or in part outside the Customer, (b) used in whole or in part by the Customer for manufacture or, in the case of computer software documentation, for preparing the same or similar computer software, or (c) used by a party other than the Customer, except for: (i) emergency repair or overhaul work only, by or for the Customer, where the item or process concerned is not otherwise reasonably available to enable timely performance of the work, provided that the release or disclosure hereof outside the Customer shall be made subject to a prohibition against further use, release or disclosure; or (ii) release to a foreign government, as the interest of the United States may require, only for information or evaluation within such government or for emergency repair or overhaul work by or for such government under the conditions of (i) above. This legend, together with the indications of the portions of this data which are subject to such limitations shall be included on any reproduction hereof which includes any part of the portions subject to such limitations.

RESTRICTED RIGHTS IN SOFTWARE

The software described in this document is licensed software and subject to **restricted rights**. The software may be used with the computer for which or with which it was acquired. The software may be used with a backup computer if the computer for which or with which it was acquired is inoperative. The software may be copied for archive or backup purposes. The software may be modified or combined with other software, subject to the provision that those portions of the derivative software incorporating restricted rights software are subject to the same restricted rights.

Copyright © 1983 Tektronix, Inc. All rights reserved. Contents of this publication may not be reproduced in any form without the written permission of Tektronix, Inc.

Products of Tektronix, Inc. and its subsidiaries are covered by U.S. and foreign patents and/or pending patents.

TEKTRONIX, TEK, SCOPE-MOBILE, and are registered trademarks of Tektronix, Inc. TELEQUIPMENT is a registered trademark of Tektronix U.K. Limited.

Printed in U.S.A. Specification and price change privileges are reserved.

GUIDE TO DOCUMENTATION

This page shows the manuals you are most likely to use with your 8560 Series system. We recommend that you acquaint yourself with each of these manuals. (You probably won't read any manual all the way through, but we do suggest that you acquire a general idea of which information is contained in which manual.) Section 1 of each manual contains pointers to the rest of the information in the manual.

This manual explains TNIX, the operating system of your 8560 Series system, and describes standard 8560 Series features.

8560 Series MUSDU System Users Manual

This manual tells how to unpack and install the 8560 Series system. It also explains the operations to be done by the system manager—the person responsible for connecting 8540s and 8550s and maintaining accounts, software, and other aspects of a multi-user system.

8560 Series MUSDU System Manager's Operation and Installation Guide

The TNIX operating system contains an online "manual page" of information about commands. You can show the information on the terminal screen, or print it on your printer.

8560 Series Online Documentation

This optional accessory manual contains printed versions of the online manual pages for standard TNIX commands. You may want to order this manual if you find yourself printing many of the online manual pages.

8560 Series MUSDU System Reference Manual

In addition to the above documentation, you may also be using manuals for other 8500 Series instruments (8540s and 8550s) or software products. The *Learning Guide* of the *8560 Series System Users Manual* contains a list of user manuals for many products used with 8560 Series systems.

Digitized by the Internet Archive
in 2024

https://archive.org/details/manualsplus_02776

ABOUT THIS MANUAL

This manual provides a complete description of all standard TNIX commands. Also described here are system calls, standard system subroutines, and other information that you may require.

Many TNIX commands are included in optional 8560 Series software packages. Those commands are not described here, but are described in the corresponding manual for the optional package.

This manual is one of several sources of information about your 8560:

- The 8560 Series System Users Manual is your guide to using the 8560 and 8561 Multi-User Software Development Units. That manual provides an overview of the 8560 system, as well as detailed information on the TNIX operating system and all standard 8560 features.

NOTE

Unless noted otherwise, all references in this manual to the 8560 refer to the 8561 as well.

- The 8560 Series System Manager's Guide shows how to install and check out your 8560 and how to configure workstations and peripherals around an 8560. If your 8560 system is not installed, refer to the System Manager's Guide before you go any further. That manual also shows how to perform system maintenance activities, such as creating user accounts, installing software, and verifying disk integrity.

- The optional 8560/8561 Service Manual contains servicing instructions for your 8560 or 8561 MUSDU. All servicing should be performed only by qualified service personnel.

This System Reference Manual is organized as follows:

Section 1. Standard TNIX Commands. This section provides detailed information about each standard TNIX command. (Optional commands are described in the manual that accompanies the optional software package.)

Section 2. System Calls. This section explains the system calls that you may use in your C programs or assembly language programs.

Section 3. Standard Subroutines. This section provides information about the standard subroutines that you may use in your C or assembly language programs.

Section 4. Special Files. This section discusses data transfer and access to/from special files. Special files are hardware input/output devices (such as terminals, printers, or disk drives) that appear to you as special files.

Section 5. File Formats. This section describes some of the data structures, file structures, and file formats that are used with the TNIX operating system and with TNIX programming tools.

Section 6. Glossary.

Section 7. Index.

CONTENTS

Page

Section 1 STANDARD TNIX COMMANDS

Introduction 1-1
Command Index 1-2
Command Page Format 1-8
Notation Conventions 1-9
Command Descriptions 1-10

Section 2 SYSTEM CALLS

Introduction 2-1
  Assembler Information 2-1
Error Messages 2-2
System Call Index 2-6
System Call Descriptions 2-9

Section 3 STANDARD SUBROUTINES

Introduction 3-1
  Subroutine Libraries 3-1
  Error Messages 3-2
  Using Subroutines in Assembly Language Programs 3-2
Subroutine Index 3-3
Subroutine Descriptions 3-7

Section 4 SPECIAL FILES

Introduction 4-1
Special File Index 4-1
Special File Descriptions 4-2

Section 5 FILE FORMATS

Introduction 5-1
File Format Index 5-1
File Format Descriptions 5-2

Section 6 GLOSSARY

Section 7 INDEX

ILLUSTRATIONS

Fig. No.		Page
1-1	An algorithm for performing incremental dumps	1-50

TABLES

Table No.		Page
1-1	**cvt** Script Files	1-31
1-2	Swap Parameters	1-33
1-3	Magnetic Reel-to-Reel Tape Block Capacities	1-51
1-4	Magnetic Cartridge Tape Block Capacities	1-52
1-5	**ed** Command Quick Reference	1-57
1-6	Search and Regular Expression Quick-Reference	1-59
1-7	Filesystem and Logical Device Block Capacities	1-120
1-8	Shell Metacharacters	1-164

Section 1

STANDARD TNIX COMMANDS

INTRODUCTION

This section describes the standard TNIX commands. The following Command Index lists these commands by function.

Many commands are included in optional 8560 Series Command packages. Those commands are **not** described here, but are described with the following packages:

- The **8560 Series Auxiliary Utilities Package** is a set of miscellaneous software utilities. The Auxiliary Utilities Package includes: pattern scanning, "computer-aided instruction" on TNIX, calculators, macro processing, simple graphics, backup media transfer, and various file manipulation utilities.

- The **8560 Series Native Programming Package** is a set of tools to develop software on the 8560. You can use these tools to create customized software tools and incorporate them into TNIX, the 8560's operating system. The following programming languages are provided: C, a system programming language; BASIC dialect, a simple applications language; and a relocatable assembler (no macros). The Native Programming Package also includes utilities for debugging, archiving, syntax checking, source formatting, stream editing, generating translators, and other software development tasks.

- The **8560 Series Text Processing Package** is a set of software tools used to prepare documentation necessary to support a development project. The Text Processing Package includes a document formatter for typewriter-like terminals or printers, and a typesetter driver, for phototypesetters. Also included are utilities for typesetting mathematical equations, building and searching reference indices, and building tabular data.

COMMAND INDEX

File and Directory Manipulation

cd Change working directory 1-17

cp Copy files 1-26

ed Invoke the standard TNIX text editor 1-55

ln Create a link to a file 1-93

ls, ll, lf, lr, lx
List contents of directory 1-100

mkdir Create a directory 1-117

mv Move or rename a file or directory 1-138

pwd Display name of current (working) directory 1-150

rm Remove (unlink) a file 1-156

rmdir Remove (unlink) a directory 1-156

tail Display the last lines of a file 1-187

Showing Files

cat Concatenate and print 1-16

lpr, lp1r, lp2r
Send to line printer 1-99

ls, ll, lf, lr, lx
List contents of directory 1-100

more Display a screenful at a time 1-132

pr Format a file 1-145

File Processing

cmp Compare two files 1-24

comm Select or reject lines common to two sorted files 1-25

diff Display differences between files 1-38

egrep Search a file for a pattern 1-75

find Find files 1-68

fgrep Search a file for a pattern 1-75

grep Search a file for a pattern 1-75

make Maintain files 1-109

sort Sort or merge files 1-169

tee Send output to a file and to standard output at the same time 1-189

touch Update last modified date of a file 1-193

tr Translate characters 1-194

uniq Report repeated lines in a file 1-202

wc Count words, lines, or characters in a file 1-206

Program Development

8540 Run commands on an 8540 1-10

8550 Run commands on an 8550 1-10

asm Invoke Tektronix Series B Assembler 1-11

atobobj Convert object code from Series A Assembler format to Series B Assembler format 1-13

libgen Invoke library generator 1-89

link Link object modules 1-91

lstr Produce linker listing 1-104

Protection and Access

chgrp Change group 1-23

chmod Change mode: add or remove read, write, execute permission 1-19

chown Change owner 1-23

login Sign on 1-95

newgrp Log in to a new group 1-139

passwd Change login password 1-143

su Switch user temporarily 1-178

Communication

mail Send mail to or receive mail from other users 1-106

mesg Permit or deny messages 1-114

who Find out who is on the system 1-207

write Send a message to another user's terminal 1-209

Status

date Show or set the date 1-35

df Show number of free blocks on disk 1-37

du Summarize disk usage 1-45

ps Show status of commands currently executing 1-147

statgpib Display GPIB status information 1-172

tty Display terminal name 1-199

Help

help Find out how to use online information 1-79

index List online manual pages available on a specific topic 1-80

man Display online manual pages about a command or topic 1-113

Terminal and Device Setup

settape Set GPIB tape drive characteristics 1-158

slp Set line printer characteristics 1-167

stty Set terminal options 1-175

tset Set terminal settings and environment variables 1-197

Data Transfer

dsc50 Perform DOS/50 functions on an 8550 disk 1-40

fbr Backup and restore files 1-64

format Write disk format 1-71

mload Upload/download TEKHEX files 1-130

od Display file in octal format (octal dump) 1-141

uload Upload/download unformatted data 1-200

TNIX Command Language

du Summarize disk usage 1-45

echo Display arguments after they have been processed by the shell 1-54

expr Evaluate arguments as an expression 1-61

false Provide truth values 1-196

kill Stop a currently running command or process 1-84

ksh Invoke Keyshell interface 1-85

nice Run a command at low priority 1-140

nohup Run a command at low priority, even if logged out 1-140

sh Invoke a shell 1-162

sleep Suspend execution for an interval 1-166

sync Execute sync system call 1-179

test Evaluate an expression (used to test conditions) 1-190

time Time a command 1-192

true Provide truth values 1-196

wait Await completion of process 1-204

System Manager Commands

(In order to use any of these system manager commands, you must be logged into the root account.)

cron Execute files at specified dates and times 1-27

cvt Examine or alter kernel 1-29

dump Backup files or file system 1-46

dumpdir Lists the names of files on a dump volume 1-53

format Formats a hard disk or flexible disk 1-71

getty Read tty characteristics 1-73

init Initialize system processes 1-81

install Install software 1-83

ldgpib Initialize the GPIB Interface 1-87

lpd, lp1d, lp2d
Control line printer spooling 1-97

makekey Generate encryption key 1-112

mkboot Copy a boot block to the hard disk in standalone mode 1-116

mkfs Construct a filesystem 1-119

mkgpib Make, modify, or print GPIB device definitions 1-122

mkgroup Add new group to system 1-127

mkuser Create or modify a user's account 1-128

mount Mount filesystems 1-136

restor Restore files from backup 1-151

setuser Set up a user account 1-161

shutdown Bring the 8560 system down in an orderly manner 1-165

syschk Check and optionally repair the file system 1-180

umount Unmount filesystems 1-136

update Periodically execute the **sync** system call 1-203

wall Write to all users 1-205

Special Keys

CTRL-C Interrupt command or program execution

CTRL-D Signal the end of input (as in **mail**) or terminate the current subshell

CTRL-H Delete input character

CTRL-K Restore last line entered, one character for each CTRL-K

CTRL-Q Resume output

CTRL-R Reprint input line

CTRL-S Suspend output

CTRL-U Delete input line

BACKSPACE Delete input character

TNIX Environment Variables

IU Currently selected port number connected to 8540 Integration Unit or 8550 MDL

uP Currently selected target microprocessor

HOME Your login directory

PATH Sequence of directories in which **sh** looks for commands

TERM Currently selected terminal type

TERMCAP	Current definition of terminal capabilities
KSH	Current invocation flags and options for **ksh** command
MORE	Current invocation flags and options for **more** command
MAIL	Pathname of incoming mail
PS1	Primary **sh** prompt string
PS2	Secondary **sh** prompt string

COMMAND PAGE FORMAT

The commands in this section are presented in the following format:

Examples	One or more demonstrations of correct entry format.
Syntax	The required syntax of the command.
Parameters	A description of the values you supply.
Explanation	A complete explanation of the command and how to use it.
Files	A list of files relevant to the command.
See Also	References to related information in this manual or in other manuals.
Notes	Information or warnings of special interest. Especially situations that require caution.
Exit Status	After a command is through executing, an exit status code is set. This subsection defines what the exit status codes are and what they mean.
Error Messages	Explanations of error messages issued by the command.

NOTATION CONVENTIONS

The following conventions are used in this section:

1. Boldface items are required and must be entered exactly as shown.

2. Underlined or italicized items represent parameters for you to enter.

3. Brackets [ ] enclose optional items.

4. Braces { } enclose an item that **must** be included.

5. Stacked items indicate that you may choose only **one** of the items.

6. Three trailing dots indicate that the immediately preceding parameter may be repeated as many times as needed.

8540, 8550 -- run commands on an 8540 or an 8550

SYNTAX
<8540/8550 command> [flag options] [argument] ...

{8540}
{8550} string

PARAMETERS

<8540/8550 command>
The name of the 8540 or 8550 command. The "<" and ">" characters are not entered.

flag options Any required command modifiers needed for the command.

argument Any required parameters needed for the 8540/8550 command.

string One or more alphanumeric characters.

EXPLANATION
The **8540/8550** command controls 8540s and/or 8550s attached to the 8560. Each command for the 8540 and the 8550 is defined on the 8560 by a link. The command names used for 8540 or 8550 commands on the 8560 are unchanged. When an 8540 or 8550 command is invoked (see the first syntax form), the **8540/8550** command formats the flag-options and arguments into a blank-separated, carriage-return-terminated command line, which it then sends to the proper 8540/8550. The **8540/8550** command then waits for the 8540 or 8550 to finish the command. The **8540/8550** command also services requests for file system operations from the 8540 or 8550.

Three "channels" are always available for the **8540/8550** command (the **TNIX** standard input, standard output, and standard error). Other "channels" may be opened and used by the 8540 or 8550 issuing an "open" request and then subsequent "read" or "write" requests.

When the **8540/8550** command is terminated, (whether by an interrupt or quit signal, or by an 8540/8550 program finishing) the **8540/8550** command saves information about any files that were opened by an HSI request (but not yet closed).

When the next **8540/8550** command requests an I/O operation that requires I/O channels other than standard input, standard output, or standard error, the **8540/8550** command looks in the file that contains the saved information. Next, the **8540/8550** command opens the file and positions the file so that it will seem as if it was never closed.

When invoked as **8540** or **8550** (see the second syntax form), string will be sent to the 8540/8550 without modification and without being prefixed by the name of the command that invoked the **8540/8550** command. All other operations will be performed as described above. This command allows 8540/8550 string assignments and 8550 command file invocations to be controlled from the 8560.

asm -- invoke the Tektronix processor-dependent assembler

EXAMPLES

asm '' 8085.l 8085.s

Assemble without generating the object code file (objectfile). Note the null string ('') in place of objectfile.

asm 8085.o '' 8085.s

Assemble without producing the listing file (listfile). Note the null string ('') in place of listfile.

asm '' '' 8085.s

Assemble without generating the object code file (objectfile) **or** the listing file (listfile). Note the null strings ('') in place of objectfile and listfile.

SYNTAX
asm objectfile listfile sourcefile...

PARAMETERS

objectfile The pathname of the file to receive the assembler-generated object code. If you do **not** want **asm** to generate an objectfile, you must specify a null string in place of objectfile by typing two single quotes ('').

listfile The pathname of the file to receive the assembler-generated listing. If you do **not** want **asm** to generate a listfile, you must specify a null string in place of listfile by typing ''.

sourcefile The pathname of the file that contains the source code **asm** will assemble.

EXPLANATION
The **asm** command invokes the Tektronix Assembler. The source code, residing in one or more files, is translated into object code (machine language), which is written to the specified file. An assembler listing is also generated and is written to the specified file.

To use the **asm** command, you must set the "uP" shell variable to the microprocessor's name. For example, to assemble source code written for the the 8085 microprocessor, first set the "uP" variable to "8085" by typing:

uP=8085; export uP

Then you may use the **asm** command. The proper assembler must be installed on your system in order for the **asm** command to work.

INVOKING PROCESSOR-DEPENDENT COMMANDS

The **asm** command is also an interface to various processor-dependent commands, such as **pas**, **ics**, **icsp**, and **pdb**, that run on the 8560. The **asm** command prefixes the processor-dependent command that you enter with **/bin/$uP**, then executes the newly created command name with the **exec** system call. The arguments that you entered with the original processor-dependent command are passed unchanged to the newly created command name. You can link files to the processor-dependent command. The file with the corresponding name, /bin/${uP}asm, will be executed.

SEE ALSO

For information on the **exec** system call, see **exec** in Section 2 of this manual.

atobobj -- convert object module from Series A format to Series B format

EXAMPLES

$ atobobj /usr/richardw/lohengrin/test2.obj

Converts the Series A format object module in file "test2.obj" into Series B format. Writes the Series B format object module into the file "test2.obj.B".

$ atobobj -o convert.obj testfile.obj

Converts the Series A format object module in file "testfile.obj" into Series B format. Writes the Series B format object module into the file "convert.obj".

SYNTAX

atobobj [**-o** outfile] infile

PARAMETERS

-o outfile Specifies the pathname of the output object file. The **atobobj** command outputs the converted object module into object file outfile.

If you omit outfile, the output object file is named the same as infile, except that the filename has the two characters ".B" appended to it. In this case, the input filename may not be longer than 12 characters. Otherwise, truncation occurs when the characters ".B" are appended.

infile The pathname of the input object file which contains an object module in Series A format.

EXPLANATION

The **atobobj** command (A to B object module format converter) reads an object module in Series A format from input object file infile, converts the object module into Series B format, then writes the converted object module into an output object file.

The **atobobj** command allows you to use object modules on your 8560 that were generated by Series A software on an 8550.

The following TEKTRONIX software products generate Series A format object modules:

- 8550 Series A 8080A/8085A Assembler
- 8550 Series A 6800/6801/6802 Assembler
- 8550 Series A Z80 Assembler
- 8550 Series A 1802 Assembler
- 8550 Series A 8048/8021/8041A/8022 Assembler
- 8550 Series A (8300AXX) Linker
- 8550 Microcomputer Development Lab Pascal 8080A/8085A Compiler

Only Series B format object modules may be used on the 8560. Since object modules generated by Series A software products are in Series A format, they cannot be used "as is" on the 8560 without first converting them into Series B format.

TRANSFERRING OBJECT FILES FROM THE 8550 TO THE 8560

To transfer object files from an 8550 disk to the 8560, you must insert the disk into the 8560's flexible disk drive. Then use the 8560's **dsc50** command to extract the appropriate files from the disk. If you don't know which files are on the 8550 disk, use the **dsc50 -t** command to obtain a directory listing. For example:

```
$ dsc50 -t <CR>
./1802/ASSEMBLER/FILE1.OBJ
./1802/ASSEMBLER/FILE2.OBJ
        .
        .
        .
```

If you know the name of the object file you want, you can "pipe" **dsc50**'s output through **fgrep** to select only those files of interest. For example:

```
$ dsc50 -t | fgrep FILE2.OBJ <CR>
./1802/ASSEMBLER/FILE2.OBJ
./8048/ASSEMBLER/FILE2.OBJ
```

Notice that the **fgrep** command prints all filenames that include the character sequence **FILE2.OBJ**.

Once you know the filename, use the **dsc50 -bx** command to transfer the object file to the 8560. For instance, to transfer the object file /1802/ASSEMBLER/FILE2.OBJ to your /usr/wolfgangm directory under the filename f2.obj, type:

```
$ dsc50 -bx 1802/ASSEMBLER/FILE2.OBJ  /usr/wolfgangm/f2.obj <CR>
```

Filename of object file on the 8550 disk. Notice that the "./" characters preceding the filename are not needed.

Destination pathname of object file on the 8560.

NOTES

- Library object modules must be extracted from the library file before they can be converted.

  On the 8550, you can use the library generator (**libgen**) to extract (copy) a library object module to an object file. You can then transfer the object file to the 8560 and use **atobobj** to convert that object file into Series B format.

  Alternatively, you can use the 8550's linker to produce a Series A format load file. The linker will extract any library object modules that are required. You can then transfer the load file to the 8560 and use **atobobj** to convert the load file into Series B format.

- The **atobobj** command cannot convert an object module created by the 8550 **SAV** command.

- Although unlikely, **atobobj** may not be able to convert some object modules created by the 8550 Pascal 8080A/8085A Compiler. If this occurs, the 8560 may issue a "**memory fault**" error message. In such a case, the object module cannot be converted. Contact your Tektronix service representative for further assistance.

cat -- concatenate and print files

EXAMPLES

cat filez Displays the contents of file filez on your terminal.

cat file1 file2 >file3

Concatenates the contents of the first two files (file1 and file2) and writes the result into file3.

cat file4 >> file3

Appends the contents of file4 to the contents of file3.

SYNTAX
cat [**-u**] [file...]

PARAMETERS

-u Causes the output to be completely unbuffered. The default is to buffer the output in 512-byte blocks, unless standard output is a terminal, in which case it is line-buffered.

file The pathname of a file to be read as input.

EXPLANATION
The **cat** command is usually used to display the contents of one or more files. If you specify two or more input files, **cat** concatenates them (in the order given in the command line) before writing its output. If the file parameter is not specified, or if the argument "-" is specified, **cat** reads from standard input.

SEE ALSO
cp
pr

NOTES
Do not redirect output to a file with the same pathname as one of the specified input files. For example:

```
$ cat a bb >a
```

erases the contents of file a **before** it can be read as input.

cd -- change working directory

EXAMPLES

cd /usr/source Changes the working directory to /usr/source.

SYNTAX
cd [directory]

PARAMETERS

directory The pathname of an existing directory. Defaults to **OME**.

EXPLANATION
The **cd** command changes your working, or current, directory to that specified by directory. The working directory is the one that most TNIX file-manipulation commands refer to by default. For example, if you use **ls** without arguments to obtain a list of files, **ls** displays the files in the current directory.

To search directory, you must have execute (search) permission for that directory. See the explanation of the **chmod** command. For example, you cannot list or read a directory owned by another user that has the following attributes (as displayed with the **ls -l** command):

```
drwx------
```

Invoking **cd** without parameters returns you to your login directory (the top of your directory tree). The **$HOME** shell variable stores the pathname for this. Thus, if your login name is **amyr**, then entering any one of the following command lines will result in the same thing:

```
cd
cd $HOME
cd /usr/amyr
```

If you want to work in a different directory temporarily, use the following procedure:

```
$ sh            ["Push" down a level in the shell.]
$ cd newdir     [Change the working directory to newdir
                 and do whatever tasks are necessary.]
$ <CTRL-D>      [Type a "control-D" to "pop" the shell
                 in order to return to the directory
                 you were originally working in.]
```

Unlike most TNIX commands, **cd** is recognized and executed by the shell; that is, the shell does not fork and execute **cd** as a new process. Because the shell executes **cd** directly, your working directory is a part of your shell "environment".

SEE ALSO
chdir
pwd
sh

chmod -- change mode: add or remove read, write, and execute permissions

EXAMPLES

chmod 200 trace.o

Changes the mode of file trace.o by enabling write permission for the owner.

chmod u+rwx,g+rw,o+r restricted

For file restricted: Enables read, write, and execution permissions for the user; enables read and write permissions for the group; and enables read permission for others. The modes are changed in the order entered.

chmod o-rwx tross.c

Changes the mode of file tross.c by disabling read, write, and execution permissions for others.

chmod = *

Takes away all permissions for all files in your current directory. Any bits set in the file creation mask will dictate which protection bits remain unchanged.

SYNTAX

chmod mode file...

```
      {a}
      {u}          {+}
      {g}          {-}
chmod {o}= [,[augo]{=}[rwxst]]... file...


           {+}
           {-}
chmod [augo]{=}[rwxst] file...


chmod [augo]= file...
```

PARAMETERS

mode The absolute mode you want to incorporate. Allows you to specify an **octal** number that represents one of the following modes:

4000	set user ID on execution
2000	set group ID on execution
1000	save image after execution (for heavily used commands)
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0070	read, write, execute (search) by group
0007	read, write, execute (search) by others

file The pathname of a file or directory.

a Indicates **all** (the same as specifying **ugo**). This includes the user, group and others. If none of the parameters **a**, **u**, **g**, or **o** are specified, **a** is the default.

u Indicates **user**. This is the owner of the directory that file resides within.

g Indicates **group**. This specifies all users in your group.

o Indicates **others**. This specifies all users not in your group.

+ Indicates that the specified permission is to be added to the file's mode.

- Indicates that the specified permission is to be removed from the file's mode.

r The **read** permission.

w The **write** permission.

x The **execute** permission.

s The **set** owner or group ID permission.

t The save image permission.

EXPLANATION

Each file or directory has certain permissions associated with it. This is called its **mode**. The mode determines who has permission to read, write, or execute the file or directory. Thus you can use **chmod** to change who (**u**ser, **g**roup, or **o**thers) has permission to **r**ead, **w**rite or **e**xecute your files. To check the current mode of a file, use the **ls -l** command. You may modify the mode of a file either absolutely or symbolically.

To modify the mode absolutely, refer to the first syntax form. This allows you to specify permissions using **octal** codes.

To modify the mode symbolically, refer to the second through fourth syntax forms.

If nothing is chosen from the selection **augo**, then **a** is the default. In this case **only**, the setting of the file creation mask is taken into account. See the definition of **umask** in section 2 of this manual. In such a case any bits **set** in the file creation mask will determine which mode bits will **NOT** change. To determine what the creation mask bits are set at, enter:

```
umask
```

and **umask** will display the bits in octal. To understand what these bits mean, convert the octal number to a **nine** digit binary number. Then study the following template:

```
--- --- ---
rwx rwx rwx
 |   |   |
 u   g   o
```

The template shows how the creation mask bits are grouped and defined. The letters **r**, **w**, **x**, **u**, **g**, and **o** are defined under the **PARAMETERS** heading. For instance, suppose the file creation mask is **20** (octal). Entering the command:

```
chmod  +rwx paba.o
```

will result in **a**ll bits being set except for the **g**roup **w**rite permissions bit. This is because **20** in octal is **000010000** in binary and note that this is the **w** bit under the group (**g**) permission bits. This bit will remain **unchanged** whether it is initially set or not.

Notice that:

+ adds the specified permission to the file's mode

- removes the specified permission

= assigns the specified permission absolutely (all other bits in that group will be reset).

To specify permissions, you may use any combination of the letters **r** (read), **w** (write), **x** (execute), **s** (set owner or group ID) and **t** (save image). Not specifying any permissions (**rwxst**) is only useful with = to remove **all** permissions.

Multiple symbolic modes must be separated by commas with **NO** spaces. Operations are performed in the order specified. The permission **s** is only useful with **u** or **g**.

SEE ALSO
For information on the **chmod** system call, see **chmod** in section 2 of this manual.

For information on the **stat** system call, see **stat** in section 2 of this manual.

For information on the **umask** system call, see **umask** in section 2 of this manual.

chown
ls

NOTES
Only a file's owner (or the superuser) may change the file's mode.

You must be logged in as root to set the save image bit (see mode **1000** or the **t** parameter).

chown, chgrp -- change owner or group

EXAMPLES

chgrp wraid *	Changes the group name of all files in your current directory to **wraid**.
chown hal ztest.o	Changes the owner name of file ztest.o to **hal**.

SYNTAX
chown owner file...

chgrp group file...

PARAMETERS

owner The login name or user ID number. This is the third item listed in the /etc/passwd file under your name.

file The pathname of a file.

group The group name or group ID number. This is the fourth item listed in the /etc/group file under your name.

EXPLANATION
The **chown** command changes the owner of the specified file(s) to that of owner. The owner may be either a decimal user ID or a login name that is listed in the password file (/etc/passwd).

The **chgrp** command changes the group ID of the specified file(s) to that of group. The group may be either a decimal group ID or a group name that is listed in the group ID file (/etc/group).

FILES
/etc/passwd
/etc/group

SEE ALSO
For information on the **chown** system call, see **chown** in section 2 of this manual.

For information on the **group** file format, see **group** in section 5 of this manual.

For information on the **passwd** file format, see **passwd** in section 5 of this manual.

NOTES:
You must be logged in as root to change the owner or group of a file.

cmp -- compare two files

EXAMPLES

cmp -s asc.o tasc.o Compares the two files and returns only the exit status code.

SYNTAX

```
    [-l]
cmp [-s] file1 file2
```

PARAMETERS

-l Print the byte number (decimal) and the differing bytes (octal) for each difference.

-s Print nothing for differing files; return exit status codes only. See the heading EXIT STATUS below.

file1 The pathname for file1. If "-" is specified, then standard input is used.

file2 The pathname for file2.

EXPLANATION

The two files are compared. By default, **cmp** makes no comment if the files are the same. If they differ, **cmp** announces the byte and line number at which the difference occurred.

SEE ALSO

comm
diff

EXIT STATUS

0 Files are identical.

1 Files are different.

2 Inaccessible or missing argument.

comm -- select or reject lines common to two sorted files

EXAMPLES

comm -12 alpha beta Prints the lines common to files alpha and beta.

comm -13 alpha beta Prints lines that are in file beta but not in file alpha.

SYNTAX
comm [-123] file1 file2

PARAMETERS

file1 The pathname of the first file.

file2 The pathname of the second file.

-12 Prints only lines common to both file1 and file2.

-23 Prints only lines that are in file1 but not in file2.

-13 Prints only lines that are in file2 but not in file1.

-123 Prints nothing.

-1 Suppresses the first column of the output.

-2 Suppresses the second column of the output.

-3 Suppresses the third column of the output.

EXPLANATION
The **comm** command reads file1 and file2 and produces a three-column output. The first column lists all lines in file1 that are not in file2; the second column lists all lines in file2 that are not in file1; and the third column lists all lines that appear in **both** files. Files file1 and file2 must be already ordered in ASCII collating sequence.

If the dash (-) is used in place of file1 or file2, standard input is used.

SEE ALSO
cmp
diff
sort
uniq

cp -- copy files

EXAMPLES

cp hrd.o tst.o

Makes a copy of file hrd.o and names it tst.o.

cp file1 file2 dir.tst

Makes a copy of files file1 and file2 and places them into directory dir.tst.

SYNTAX
cp file1 file2

cp file... directory

PARAMETERS

file	The pathname of a file.
directory	The pathname of a directory

EXPLANATION
The **cp** command is used to make copies of files and rename them and/or transfer them to another directory.

In the first syntax form, **cp** makes a copy of the specified file (file1) and then renames it file2. If file2 already exists, its contents are replaced by the contents of file1. The mode and owner of file2 are preserved if it already existed; the mode and owner of the source file (file1) is used otherwise.

In the second syntax form, the specified files are copied into the directory specified by directory. The names of the files are unchanged.

SEE ALSO
cat
mv
pr

NOTES
The **cp** command will not copy a file onto itself.

cron -- execute files at specified dates and times

SYNTAX
/etc/cron

EXPLANATION
The system program, **cron**, automatically executes commands at specified dates and times. The dates, times and the commands to be executed are specified in the /usr/lib/crontab file. Now **cron** is executed only once -- at the time the system is booted. The **cron** program never exits, instead, it remains in the background as long as the system is up. The **cron** program is started by the command:

/etc/cron

in the system initialization file /etc/rc. (See **init**). This command is automatically placed in the /etc/rc file when **cron** is installed.

Crontab Syntax

Information stored in the crontab file consists of lines of six fields each. The fields are separated by spaces or tabs. The syntax of each line:

minite hour day-of-month month day-of-week command

Where:

minute
An integer in the range 0--59

hour An integer in the range 0--23.

day-of-the-month
An integer in the range 1--31.

month
An integer in the range 1--12.

day-of-week
An integer in the range 1--7. 1=Monday, 2=Tuesday, etc.

command
The command that the shell executes at the date and time specified by the beginning of the line. A percent character ("%") in this field is translated to a newline character. Only the first line (up to a "%" or end-of-line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

Each of the first five fields may also contain:

1. Two integers (in the proper range) separated by a minus ("-") indicating an inclusive range. For example, "2-4" in the second field indicates the hours 2:00 through 4:00 (AM).

2. A list of integers, separated by commas, indicating execution at those values only. For example, "6,17,21" in the third field indicates execution on the 6th, 17th and 21st days of the month.

3. An asterisk ("*") indicates all legal values.

The **cron** program examines the crontab file every minute.

FILES
/usr/lib/crontab

cvt -- examine or alter kernel parameters

EXAMPLES

```
fbr filename to boot>cvt
cvt arguments: -w
rootdev 0 8
rootdev 0 8
swapdev 0 8
swapdev 0 8
pipedev 0 8
pipedev 0 8
nswap 4576
nswap 4576
swplo 134576
swplo 134576
^D
```

In this example, the **cvt** command (on the Standalone Utilities disk) is used to reconfigure the TNIX kernel code in /tnix to accommodate a root filesystem mounted on two fixed disks.

```
# cvt -w -f newtnix root01
```

In this example, the standard TNIX **cvt** command reconfigures the TNIX kernel code in /newtnix to accommodate a root filesystem mounted on two fixed disks. (The example assumes that /newtnix was previously copied from /tnix.) The file root01 is a file located in the /etc/cvtscript directory that contains the necessary configuration parameters. Once you are sure that /newtnix is properly configured, move /newtnix to /tnix and reboot your system.

```
# cvt -w tzflag 1
```

In this example, the standard TNIX **cvt** command informs TNIX that daylight savings time is in effect.

```
# cvt
*
```

In this example, the standard TNIX **cvt** command displays the current settings of the internal TNIX parameters.

SYNTAX

cvt [**-w**] [**-f** tnixname] [script]

PARAMETERS

-w Write the new parameters. Without the -w parameter, **cvt** can only be used to examine the current TNIX configuration or check that the new parameters are compatible with the current system hardware configuration. -w establishes the new parameters within TNIX.

-f tnixname
Examine or modify tnixname, rather than the default kernel (**/tnix**).

script A file that contains lines directing **cvt** to examine or set various configuration parameters. If the file you specify for script cannot be found in the current directory and does not begin with a slash, **cvt** checks for script in the /etc/cvtscript directory. If you don't specify script, **cvt** takes its input from the standard input. (If you use standard input, terminate **cvt** by typing a CTRL-D.)

Within the script file, blank lines and lines beginning with a pound-sign (#) are ignored. Lines beginning with an exclamation point (!) are passed to the shell for execution. All other lines must be of the form:

parameter

or

parameter value

where parameter is the name of the parameter to be set or examined, and value (if specified) is the new value for the parameter. If you don't specify value, **cvt** prints the current value of parameter.

NOTE

Because **cvt** deals with the kernel's low-level configuration parameters, it is possible to generate a kernel that will not run properly. The use of configuration scripts is encouraged.

EXPLANATION

The **cvt** command is named after an internal TNIX data structure called the "Configuration Variable Table", and is used to alter three sets of TNIX kernel parameters:

- Data structure size parameters. These parameters establish sizes for various data structures within the kernel.
- Root device parameters. These parameters describe the root filesystem configuration, and must be altered whenever you change the size of the root filesystem. (The root filesystem always begins on the 8560's internal fixed disk). The /etc/cvtscript directory contains script files that specify these parameters for each possible 8560 root filesystem configuration. Table 1-1 identifies these script files.

- Time parameters. These parameters enable TNIX to properly compute local times and dates.

Table 1-1
cvt Script Files

Root Filesystem Name	Script File Name	Number of Disk Drives
/dev/hd0	root0	1
/dev/hd01	root01	2
/dev/hd02	root02	3
/dev/hd03	root03	4

Data Structure Size Parameters

buffer — The total number of I/O buffers. Most disk I/O occurs indirectly through these buffers. The default value for this parameter is 20. If you have enough memory, you can increase the value up to 30. Having more buffers increases the disk I/O throughput, thereby increasing the performance of disk-intensive programs. However, the search time for all programs will also increase.

inode — The total number of active inode table entries. Each table entry identifies a unique, active (open) file. The default value for inode is 62. If your system often reports "file table overflow" or "inode table overflow" errors, you should increase this value if you have enough memory. If you increase this parameter, you must also increase the file parameter proportionally.

file — The total number of open file table entries. This parameter is similar to the inode parameter, except that table entries occur for each instance that a file is opened. For example, if three independent processes opened the same file, there would be one entry in the active inode table and three entries in the active file entry. (However, if the processes were forked from a common parent, they will share a common open file table entry.) The default value for file is 80; the value must always be greater than or equal to the inode parameter value.

proc — The total number of process table entries. This table contains one entry for each process. Each entry contains all the information the system needs about a process while the process is swapped out of primary memory. The size of this table determines how many processes may exist within the system at one time. The default value for proc is 50. The value must always be greater

than or equal to the value for the text parameter, and less than or equal to the value for the swapmap parameter.

mount — The maximum number of mount table entries. This table contains one entry for each mounted device. (TNIX expects the mount table to include the root device, even though the **mount** and **umount** commands do not affect the root device.) The default value for mount is 4. However, you can decrease this value to gain a little extra memory space if you have fewer than 3 mountable (non-root) filesystems. The parameter must always be 1 or greater.

text — The maximum number of text table entries. This table contains one entry for each current read-only text (code) segment. Each entry contains a segment's address in primary and secondary memory, and a reference count that indicates how many processes share the segment. The default value for proc is 25; the value must always be less than or equal to the value for the proc parameter.

swapmap — The maximum number of swap map entries. Each entry identifies a discontiguous section of the swap space (located on the root device). Therefore, the parameter sets the maximum fragmentation of the swap space. The default value for swapmap is 60; the value must always be greater than or equal to the value for the proc parameter.

coremap — The maximum number of core (primary memory) map entries. Each entry identifies a discontiguous section of available user memory. Therefore, the parameter sets the maximum fragmentation of the memory. The default value for coremap is 60.

Root Device Parameters

rootdev — Two integers (separated by a tab or space character) that specify the major and minor device numbers of the root device. The major device number **must** be 0. The minor device number **must** be 0 (for /dev/hd0), 8 (for /dev/hd01), 16 (for /dev/hd02), or 24 (for /dev/hd03).

swapdev — The major and minor device numbers of the swap device. These values must be identical to the values of the rootdev parameter, since TNIX allows the swap space to exist only on the root device.

pipedev — The major and minor device numbers of the pipe device. These values must be identical to the values of the rootdev parameter.

nswap — The number of blocks allocated for swap space. Table 1-2 shows the default values for each size of root filesystem. If you want to increase the space available for swapping, increase the nswap parameter and decrease the swplo parameter making the total of nswap and swplo equal the total number of blocks available on the device.

swplo The starting block of the swap space on the root device. This block number marks the boundary between the root filesystem and the swap space. Make sure the total of nswap and swplo equals the total number of blocks available on the device.

CAUTION

Be sure to do a complete dump of the root filesystem before changing the nswap and swplo parameters. A larger swap space may overrun existing files. After you have modified these parameters, execute **mkfs**, setting the **mkfs** blocks parameter to the value of the swplo parameter. Then restore your root filesystem.

Table 1-2
Swap Parameters

Root Device Size	Device Name	Total Blocks	Default swplo	Default nswap
13.6M Bytes	/dev/hd0	26610	24360	2250
35.6M Bytes	/dev/hd0	69576	65000	4576
71.6M Bytes	/dev/hd01	139152	134576	4576
106.8M Bytes	/dev/hd02	208728	204152	4576
142.4M Bytes	/dev/hd03	139152	273728	4576

Time Parameters

timezone The offset in minutes from Greenwich mean time to the current time zone. For example, if you are in the Pacific standard time zone, the value for timezone is 480.

tzname The three-character name of the current time zone. For example, EST means Eastern standard time.

tzdstname The three-character name of the current time zone during daylight savings time. For example, PDT means Pacific daylight time.

tzflag Indicates whether daylight savings time is in effect. The value for tzflag may be "1" for daylight savings time, or "0" for standard time.

Using cvt

Use the Standalone Utilities version of **cvt** when you change the configuration of the root filesystem. Use the standard version of **cvt** in other circumstances.

Shut down and reboot the 8560 after you reconfigure /tnix so that the new TNIX will be loaded into memory.

FILES

/tnix The default tnix kernel.

/etc/cvtscript The directory that contains distributed **cvt** scripts.

SEE ALSO

8560 Series System Managers Guide

For information on the Configuration Variable Table format, see **dump** in Section 5 of this manual.

ERROR MESSAGES

preposterous cvt address
Tnixname is not in the appropriate format.

SYSTEM TOO BIG: number BYTES OVER
The new configuration will not fit into the available memory. Try reducing one of the data structure size parameters or installing additional memory in your system.

Other messages are self-explanatory.

date -- show or set the date

EXAMPLES

date 14-nov-83 12:45

Set the date to Nov 14, 1983 12:45 PM.

date 3:45

Sets the time to 3:45 AM.

date -t PST -d PDT -w 8:30

Sets the time zone strings to PST and PDT, and the offset from Greenwich mean time (GMT) to 8 1/2 hours.

SYNTAX

date [dd-mmm-yy] [hh:mm[:ss]] [**-t** SSS] [**-d** DDD]
[[**-w** [hh:mm]] [**-e** [hh:mm]]]

date [[yy]mmddhhmm [.ss]] [**-t** SSS] [**-d** DDD]
[[**-w** [hh:mm]] [**-e** [hh:mm]]]

PARAMETERS

dd — The day of the month, specified by entering a number in the range 01--31.

mmm — The month, specified by entering the first three lowercase letters of the name of the month. For example, **jul** would specify the month of **July**.

yy — The year, specified by entering the last two digits of the year.

hh — The hour (24-hour system), specified by entering a number in the range 0--24.

mm — The minute, specified by entering a number in the range 00--59.

ss — The second, specified by entering a number in the range 00--59.

mmddhhmm — An 8-digit number representing the month (the leftmost mm, 00--12), day (dd, 01--31), hour (hh, 00--24), and minute (the rightmost mm, 00--59).

-t SSS — Defines the system standard time zone string variable (SSS). If this string is not set or is null, the system prints out GMT +/- HH for the time zone. HH represents the number of hours west of GMT.

-d DDD Defines the system daylight time zone string variable (DDD). If this string is null or if it was never set, the system assumes that there is no daylight savings time.

-w hh:mm Sets the hours and minutes west of Greenwich mean time (GMT) that the time zone strings represent if you are west of the Prime Meridian.

-e hh:mm Sets the hours and minutes east of Greenwich mean time (GMT) that the time zone strings represent if you are east of the Prime Meridian.

The preceding parameters may be invoked independently of setting the date, time, or other parameters.

EXPLANATION

The **date** command is used to set the date and time. If no argument is given, the current date and time are printed.

If only the time is set, the day-month-year string is not modified. If only the day-month-year string is specified, the time remains unchanged. Also, if the seconds parameter (ss) is omitted, system seconds are not modified.

The **date** command takes care of the conversion to and from local standard and daylight time in the United States.

ERROR MESSAGES

No permission You must be a superuser to use the **date** command to modify the date or time.

bad conversion The date set is syntactically incorrect.

df -- show number of free blocks on disk

EXAMPLES

df /dev/hd1 Prints the the device name (/dev/hd1) and the number of free blocks available for file allocation in the associated filesystem.

SYNTAX

df [device-name...]

PARAMETERS

device-name The device name that the filesystem starts on.

EXPLANATION
The **df** command prints out the number of free blocks available for file allocation on a specific **filesystem.** If no device name is specified, the free space on each filesystem listed in the file /etc/checklist is printed. If **df** cannot read /etc/checklist, the /dev/rhd0 filesystem will be used.

To use the **df** command, you only need to know the name of the disk that the filesystem starts on, since a filesystem can span more than one device.

If you don't know the current filesystem names and their associated device names, use the **mount** command. As an example:

```
$ /etc/mount
hd1 on /usr1
```

In this case, there is a filesystem named /usr1 on device /dev/hd1.

FILES
/etc/checklist

SEE ALSO
For more detailed information on filesystems, see the **8503 Disk Expansion Unit Users Manual.**

For information on the **checklist** file format, see **checklist** in Section 5 of this manual.

For information on the **hd** file, see **hd** in Section 4 of this manual.

mount

diff -- display differences between files

EXAMPLES

diff f1 f2

Shows the lines that differ between files **f1** and **f2**.

diff -e outp.bak outp.asm >edcmds

Produces a script of **ed** commands that tell how to change outp.asm into a duplicate of outp.bak.

SYNTAX

```
     [-e]
     [-f]
diff [-h][-b] file1 file2
```

PARAMETERS

-b Causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

-e Produces a script of **a**, **c** and **d** commands for the editor (**ed**), which recreates file2 from file1.

-f Produces a script similar to that produced by **-e**, but in the opposite order. The script produced is not useful for the **ed** command.

-h Does a fast, simplistic job. It works only when changed stretches are short and well separated, but does work on files of unlimited length.

EXPLANATION

The **diff** command compares two files and tells what must be changed to bring them into agreement. If file1 or file2 is specified by a dash (-), standard input is used. If file1 (file2) is a directory, then a file in that directory whose filename is the same as the filename of file2 (file1) is used. Normally, output consists of lines in the following forms:

n1 a n3,n4

n1,n2 d n3

n1,n2 c n3,n4

These lines resemble **ed** commands to convert file1 into file2. The **-e** parameter produces actual editor commands for use in converting files. The numbers after the letters pertain to file2. In fact, by exchanging "a" for "d" and reading backward you may determine how to convert file2 into file1. As in **ed**, identical pairs where n1 = n2 or n3 = n4 are abbreviated as a

single number.

Following each of these lines come all the lines that are affected in the first file flagged by "<", then all the lines that are affected in the second file flagged by ">".

Except in rare circumstances, diff finds a smallest sufficient set of file differences.

FILES

/tmp/d?????

/usr/lib/diffh Used with the **-h** parameter.

SEE ALSO

See the TNIX Editor ("ed") section of the **8560 Series System Users Manual**

cmp
comm
ed

NOTES

Editing scripts produced under the **-e** or **-f** parameter will not correctly create lines consisting of a single period.

EXIT STATUS

0 No differences occurred.

1 Some differences occurred.

2 An error occurred.

dsc50, mk50 (make), **wr50** (write), **del50** (delete), **rd50, dir50** (table of contents), **ln50** (link) -- perform DOS/50 functions on an 8550 disk

SYNTAX

dsc50 [**-mwdxtls**] [**-epbrvq**] [**-c** path] [**-o** owner]
[**-n** name] [**-a** n] [**-f** disk] [source... dest]

EXAMPLES

dsc50 -t | fgrep TESTS

Prints out only the lines in the directory listing of an 8550 diskette that contain the character sequence **TESTS.** Each line consists of the pathname for each directory or file, preceded by the two characters, "./". When specifying the **dest** or **source** parameters, you may ignore the "./" characters.

dsc50 -xb MAIN/TESTS/input.obj /usr/alenb

Copies (extracts) file input.obj **from** an 8550 diskette (in directory MAIN/TESTS) **to** directory /usr/alenb on the 8560.

Object files produced by assemblers or compilers must **always** be transferred using the **-b** parameter.

dsc50 -xb MAIN/TESTS /usr/alenb

Copies (extracts) **all** files in directory MAIN/TESTS **from** an 8550 diskette **to** the 8560 under the directory /usr/alenb. Files within subdirectories are ignored.

dsc50 -xbr MAIN/TESTS /usr/alenb

Copies (extracts) **all** files in directory MAIN/TESTS (and **all** files in each of its subdirectories) **from** an 8550 diskette **to** the 8560 under the directory /usr/alenb.

dsc50 -w /usr/alenb MAIN/TESTS

Copies (writes) **all** files in directory usr/alenb **from** the 8560 **to** an 8550 diskette under the directory MAIN/TESTS. Files within subdirectories are ignored.

dsc50 -wbr /usr/alenb MAIN/TESTS

Copies (writes) **all** files in directory usr/alenb (and **all** files in each of its subdirectories) **from** the 8560 **to** an 8550 diskette under the directory MAIN/TESTS. For example, if the file /usr/alenb/directory/2.obj existed, it would be written on the 8550 diskette under the pathname MAIN/TESTS/directory/2.obj.

dsc50

If no parameters are specified, **dsc50** defaults to interactive mode. See the explanation under "Interactive Mode".

PARAMETERS

The following parameters may be specified on the command line or in interactive mode.

source The pathname of the **source** directory or file.

dest The pathname of the **destination** directory or file.

-m **m**ake -- Initializes (makes) a new DOS/50 volume. Similar to the **-w** parameter, except that the 8550 diskette is initialized (without physically formatting it) **before** copying the file(s).

-w **w**rite -- Copies the specified TNIX source file(s) to pathname dest onto an 8550 diskette. A source filename of "-" refers to standard input (usually your terminal).

-d **d**elete -- Deletes the specified 8550 source files from the 8550 diskette.

-x **x**tract -- Copies the specified 8550 source file(s) from an 8550 diskette to the specified TNIX destination file/directory (dest). A dest filename of "-" refers to standard output.

-t **t**able of contents -- Prints a directory listing of the source file(s). If no source is specified, the current directory is used.

-l **l**ink -- Links the existing 8550 source file(s) to the 8550 dest file/directory.

-s **s**etup -- Prints the **setup** (the current state of all parameters) to standard output in the form of a **dsc50** command. This is useful only in **interactive** mode because it helps you see what parameters you have set.

-c path **c**hange path -- Changes the current 8550 directory to path. This will be the directory relative to which all 8550 pathnames not beginning with slash (/) are interpreted. An error is reported if directory path does not exist. The default current directory is "/", the root directory of the 8550 diskette.

-o owner owner -- Specifies the owner name given to anything created on the 8550 diskette (files, directories, or the entire volume). The default owner name is "NO.NAME".

-n name name -- The volume name given to the 8550 diskette when it is created by the -m operation. The default name is "NO.NAME".

-a n amount -- Specifies the number (amount) of file slots to be created on the 8550 diskette by the -m operation. This means that n different files may be stored on the disk, up to a maximum of 256 files (the default amount).

-f disk file -- Redirects source file input to a TNIX file rather than the default device, /dev/rfd0 (the flexible disk drive on the 8560).

The following parameters may be used on the command line or in interactive mode. In interactive mode, specification of one of these parameters toggles the effect. That is, entering the parameter the first time activates it and entering it again will deactivate it.

-e exit -- Terminates interactive mode after executing the given operation(s). This is valid only in interactive mode.

-p prompt -- In interactive mode, **dsc50** prompts for each command with a dash (-). This is valid only in interactive mode. The default in interactive mode is **no prompt**.

-b binary transfer -- Because the "end of line" character differs between TNIX and DOS/50 text files, some character translation must occur when a text file is copied to or from the disk. This translation is performed by **dsc50** on all files **unless** the -b parameter is specified.

-r recursive -- If any directory appears in a source directory, the current command is applied recursively to that subdirectory. Normally, subdirectories are skipped.

-v verbose -- Normally **dsc50** works silently. The -v (verbose) parameter causes it to type the name of each file it treats, preceded by the operation name. With the -t (table of contents) operation, -v prints the following information about each file:

- file type and capabilities of the file
- modification date-time
- owner's name
- size
- link count

-q query. Causes **dsc50** to pause before processing each file, display the operation name and the filename (as with **-v**), and await your response. When you respond with **y** or **Y**, the file is processed. When you respond with **n** or press the RETURN key, the file is not processed. A response of **N** is the same as answering **n** to this and all further questions involved in the current operation. A response of **Y** is the same as answering **y** to this and all further questions involved in the current operation. Entering a **<CTRL-D>** (End-of-file) on standard input will terminate the operation. Query does not take place for the table of contents (**-t**) or setup (**-s**) operations.

EXPLANATION

The **dsc50** command performs many DOS/50 functions on an 8550 diskette. The operation to perform can be specified as a command line option in interactive mode, or as the name of the command.

Command Line Mode

Specifying an operation in the command name is equivalent to invoking **dsc50** with the corresponding parameter. For example:

```
$ wr50 /usr/daveh/tset /SOURCES/SETUP
```

This is the same as typing:

```
$ dsc50 -w /usr/daveh/tset /SOURCES/SETUP
```

In 8550 pathnames, "." refers to the current directory, ".." to its parent directory (see the **-c** parameter).

Any directory appearing as a source file specifies all files within that directory (see the **-r** parameter). Source subdirectories and inaccessible files are ignored, unless you have specified the **-r** or **-t** parameters.

If more than one source file is specified, or if source is a directory, dest must be a directory.

If dest is a directory, the source file(s) will be copied into, or linked with, the corresponding file(s) within that directory. For example, if /mystuff is a directory on the 8550 diskette,

```
$ dsc50 -w /usr/daveh/adb.hex /usr/src/sccs.src /mystuff
```

copies files /mystuff/adb.hex and /mystuff/sccs.src into the 8550 diskette. If the dest file already exists, it is overwritten during copy operations. During link operations, if the dest file already exists, it is unlinked.

Interactive Mode

Interactive mode is entered by invoking **dsc50** with no parameters. Once in this mode, dsc50 reads commands from standard input until the **-e** parameter is specified or the end-of-file is encountered. Commands in interactive mode have the same syntax as **dsc50** with the following exceptions:

- The command name is omitted.
- An initial "-" for each parameter is optional.
- Any line beginning with an exclamation mark (**!**) is given to **sh** to be interpreted as a shell command.

TRANSFERRING FILE HIERARCHIES

You can transfer an entire file hierarchy, including source and object files, from an 8550 to an 8560 using the recursive (**-r**) and binary (**-b**) parameters:

```
$ dsc50 -rb source.dir dest.dir
```

Here, source.dir is the source directory and dest.dir is the destination directory. However, source or ASCII text files transferred in this manner are not properly converted. The end-of-line character is a **carriage return** (015 octal) for an 8550 text file, and a **newline** (012 octal) for an 8560. All ASCII files transferred with the recursive (**-r**) and binary transfer (**-b**) parameters set for **dsc50** must be processed so that the end-of-line character is changed from a **carriage return** to a **newline.** To do this, first put a list of the files to be transferred into the file temp.list:

```
$ find dest.dir -type f -print > temp.list
```

Manually edit the temp.list file to remove the names of all files that should not be processed, such as object files. Then use the file as input for a shell script that translates **carriage return** characters to **newline** characters:

```
$ cat temp.list |
> while read x
> do tr "\015" "\012" < $x > tmp$$
> mv tmp$$ $x
> done
```

NOTES

Interrupting **dsc50** while it is modifying the DOS/50 diskette (For example, during a make, write, delete, or link operation.) may cause corruption of that diskette.

Because **dsc50** deals with only one 8550 diskette at a time, 8550 pathnames are not prefixed by /VOL/volume.name.

du -- summarize disk usage

EXAMPLES

du -s /usr/lorii Prints the number of blocks stored under the directory name /usr/lorii.

SYNTAX

du [**-a**] [**-s**] [file...]

PARAMETERS

-a Causes an entry to be generated for each file.

-s Causes only the grand total of each file or directory specified to be given.

file The pathname of a file or directory.

EXPLANATION

The **du** command reports the number of 512-byte blocks required to store the specified file(s) and/or directory(s) on disk.

When specifying a directory, any files in subdirectories within the specified directory(s) will be counted also. That is, **du** will perform recursively.

If file is missing, '.' is used.

Absence of arguments causes an entry to be generated for each directory within your current directory only.

A file that has two links to it is only counted once.

NOTES

Non-directories given as arguments (not under the **-a** parameter) are not listed.
If there are too many distinct linked files, du counts the excess files more than once.

dump -- backup files or filesystem

EXAMPLES

To perform a complete dump of the root filesystem onto flexible disks, updating /etc/ddate, type:

```
# dump 0u /dev/rhd0 <CR>
```

To perform a complete dump of the non-root filesystem /dev/rhd23 onto flexible disks, updating /etc/ddate, type:

```
# dump 0u /dev/rhd23 <CR>
```

To do a full dump of /dev/rhd1, updating /etc/ddate, onto 10,000-block tapes (using /dev/mt0), verifying each volume before and after it is written, type:

```
# dump 0u /dev/rhd1 -f /dev/mt0 -s 10000 -v <CR>
```

To dump all files on /dev/rhd0 that have changed since the last level 0 dump, updating /etc/ddate, onto 20,000-block tapes (using /dev/mt0), verifying each volume before it is written, type:

```
# dump 1u /dev/rhd0 -f /dev/mt0 -s 20000 -m <CR>
```

To dump all files on /dev/rhd0 that have changes since the last level 7 or earlier dump onto flexible disks, updating /etc/ddate, type:

```
# dump 8u /dev/rhd0 <CR>
```

To do a full dump of the filesystem on /dev/rhd0, updating /etc/ddate, onto single-sided, single-density flexible disks, type:

```
# dump 0u /dev/rhd0 -s 500 <CR>
```

SYNTAX

dump [level[u] logical-device] [**-f** dumpdev] [**-s** block-size] [**-m**] [**-v**]

PARAMETERS

u — If the dump completes successfully, writes the date of the beginning of the dump into the /etc/ddate file. This file records a separate date for each logical device and for each **dump** level.

level — A single, decimal digit (0 -- 9) that specifies the "dump level". All files modified since the last dump of a lesser level from a logical-device, as determined by the date stored in the file /etc/ddate, are dumped. If no date is associated with a level, the option **0** causes the entire logical device to be dumped.

All files in logical-device that have been modified since the last lesser-level dump (determined by the date stored in file /etc/ddate) are dumped.

logical-device — Specifies the pathname of the device containing the filesystem you want to dump. The pathname must begin with /dev/. For example, if the root filesystem is located on the 8560's internal disk, then the logical device name of the root filesystem is /dev/rhd0.

-f dumpdev — Specifies the complete pathname of the dump device (the device upon which the dump will be written). The pathname must begin with /dev/. If you don't specify dumpdev, **dump** dumps to /dev/rfd0 (the flexible disk drive).

-s block-size — Specifies the size (in blocks) of the dump medium. If you don't specify size, **dump** assumes that you're dumping to 1995-block volumes.

-m — Causes **dump** to write and read back a test record on the target medium before attempting a dump. This parameter gives you the opportunity to replace a bad or improperly mounted volume without restarting the dump procedure.

-v — Causes **dump** to verify that each volume can be read, after the volume is completely written. When you include **-v** in the command line, **dump** automatically includes the **-m** parameter as well.

Using **dump** without specifying any arguments is equivalent to typing:

```
# dump 9u /dev/rhd0 -f /dev/rfd0 -s 1995 <CR>
```

EXPLANATION
The **dump** command copies all files on logical-device that have changed after a certain date to flexible disk(s) or magnetic tape(s).

The **dump** command tells you the current date, the earliest date the dump starts at (the earliest possible date if this is a level 0 dump), the devices being dumped from and to, and when each of the four internal steps of the dump procedure begin. If the dump requires more than one volume (a single flexible disk or magnetic tape), **dump** will ask you to change volumes. After changing the volume, press the RETURN key. The first volume should be installed before you invoke **dump**.

Performing A Full Backup

This procedure tells how to create a full backup of your system, referred to as a "level-0 dump", using double-sided, double-density flexible disks (dump allows you to use single-density flexible disks, magnetic tapes, and magnetic cartridges by specifying the device that **dump** writes to and/or the maximum number of blocks on each dump volume).

1. Log into the "root" account on the system console:

```
login: root
       [you may be asked for a password]
```

2. Make sure that you are the only person logged into your system.
3. Determine how many formatted, double-density, double-sided flexible disks you will need in order to perform a full backup of your system. To do this, open the flexible disk drive's door, then enter the following **dump** command to determine how many disks you will need:

```
# dump 0 /dev/rhd0

        [ several messages are displayed before dump
          prints the following message ]

dump: estimated 16259 blocks on 9 volume(s)

        [ according to this example, you should have
          9 formatted disks available before
          starting the backup procedure ]
              .
              .
              .
        [ several error messages are displayed because the
          flexible disk drive door is open ]
```

(To format a flexible disk, cover the disk's write-protect slot, insert the disk into the disk drive, then type the **format** command.)

4. To start the backup procedure, insert a formatted disk into the disk drive, then type the following command:

```
# dump 0u /dev/rhd0
```

The **dump** command will guide you through the backup process, asking you to insert new disks into the disk drive when necessary. If an additional disk is necessary, **dump** prints the following message:

```
dump: change volumes, current inode = XXX
          [ XXX is the starting inode number
            of the next volume. ]
```

Before inserting the new disk into the disk drive, label it with the following information:

- date
- dump level (0)
- volume number (the first disk is volume 1, the second is volume 2, and so on)
- starting inode number (the starting inode number for volume 1 is "0")

Once you have labeled the disk with the above information, insert it into the disk drive, close the drive's door, then press the RETURN key.

Performing Incremental Dumps

The 10 dump levels allow you to perform a complete backup of your filesystem with a level 0 dump, followed by fast, incremental dumps (levels 1--9) that store only those files that have changed since the last backup (incremental or complete).

When performing incremental backups, the rule of thumb is to perform the incremental backup at the same dump level until the incremental backup at that dump level takes longer than 15 minutes, or more than one dump volume (one flexible diskette, for example). At that point, proceed to the next higher dump level. Thus, a level 1 dump is followed by either a level 1 or level 2 dump; a level 2 dump is always followed by a level 2 or level 3 dump, and so on up to level 9. When a level 9 dump takes longer than 15 minutes or more than one dump volume, you perform a complete backup (a level 0 dump), and start over again.

Let's assume that your company has a corporate archive policy that requires you to perform a complete backup of the data on your system at the beginning of each month for permanent, off-line storage. Let's also assume that you want a fast method of backing up your system once each day, so that you never have to worry about losing more than one day's worth of data. Figure 1 presents an algorithm for performing an incremental backup. Let's look at each step:

1. At the beginning of each month, perform a level 0 dump of your entire system. The level 0 dump copies all data on a single filesystem onto the backup media (flexible disk or magnetic tape).

2. On the next day, perform an incremental backup using dump level 1. This dump copies all files that have changed since the last level 0 dump onto the backup media.

3. On the next day, perform an incremental backup at the same dump level as the last incremental backup that you performed. This dump copies all files that have changed since the last incremental backup performed at a lesser dump level. Repeat this step each day until the

incremental backup takes longer than 15 minutes or more than one dump volume to complete. Then advance to step 4.

4. If have been performing a level 9 dump, perform a level 0 dump, then return to step 2. If you have not been performing a level 9 dump, perform the next level dump, then return to step 3.

Continue the above process until the beginning of the next month, when when you start from step 1 again. Figure 1-1 illustrates this process:

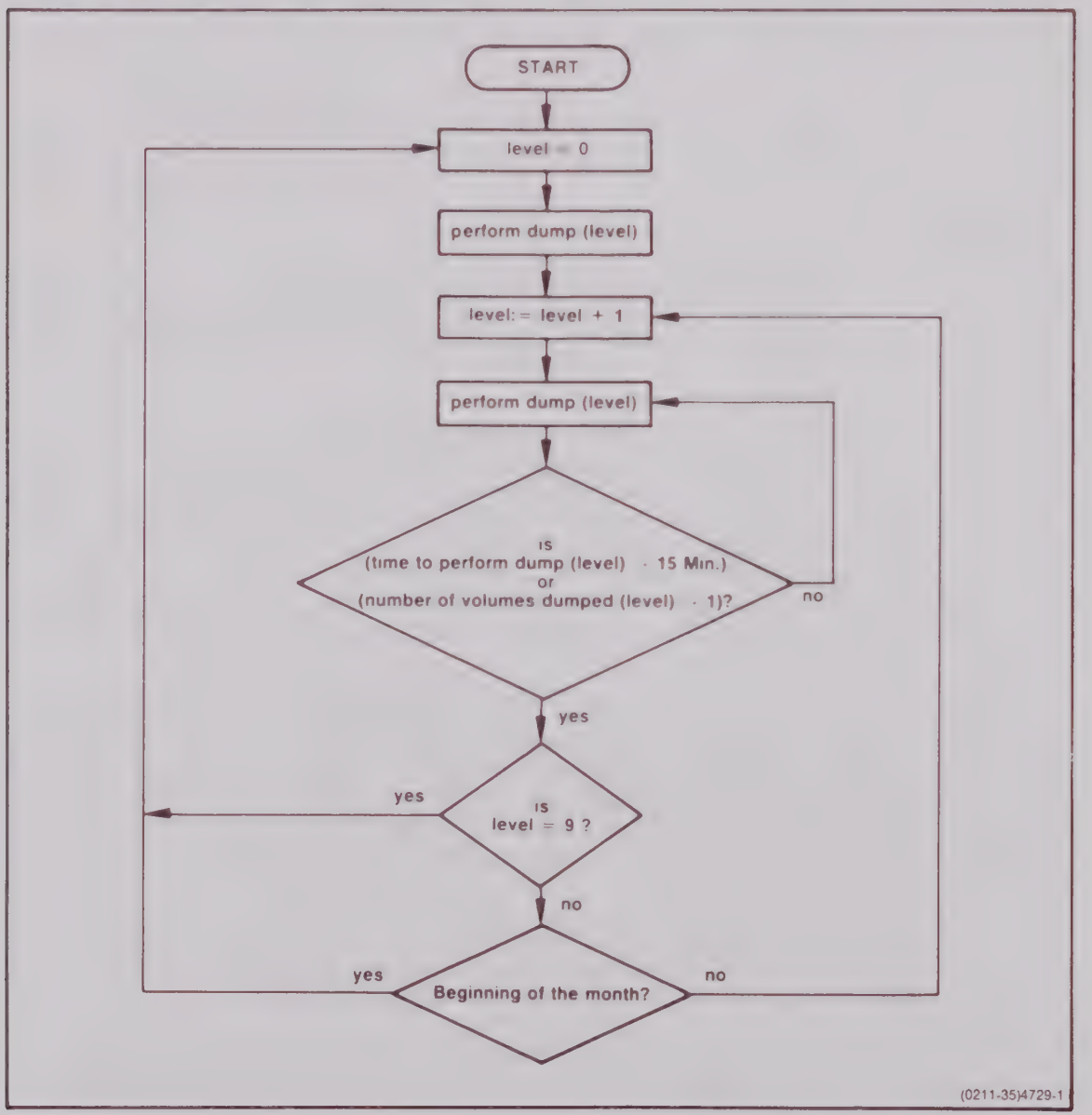


Figure 1-1. An algorithm for performing incremental dumps.

FILES

/etc/ddate -- records dump dates of filesystem/level.

SEE ALSO
8560 Series System Managers Guide

For information on the **dump** file format, see **dump** in Section 5 of this manual.

dumpdir
restor

NOTES
Read errors on logical-device are ignored. Write errors on the dump volume(s) are usually fatal.

Single-density flexible disks have a 500-block capacity. Double-density flexible disks have a 1995 block capacity.

A magnetic tape's block capacity depends on the tape length (in feet) and the **blocking factor** (the number of blocks-per-record). The following two tables show the approximate maximum block capacities **dump** allows for different tape lengths (to compensate for varying tape lengths, use a smaller number):

Table 1-3
Magnetic Reel-to-Reel Tape Block Capacities

Blocking Factor	Tape Length (Feet) 600	1200	2400
8	18016	35128	70944
7	17125	34087	68740
6	16469	32780	66106
5 (a)	15930	30940	62740
4	14521	28903	58287
3	12985	25846	52122
2	10718	21333	43021
1	7034	14000	28339

(a) **dump** uses a blocking factor of 5.

Table 1-4
Magnetic Cartridge Tape Block Capacities

Blocking Factor	Tape Length (Feet) 300	450
8	5370	8060
7	5320	7980
6	5280	7920
5 (a)	5180	7770
4	5080	7630
3	4880	7320
2	4550	6820
1	3770	5650

(a) dump uses a blocking factor of 5.

dumpdir -- list the names of files on a dump volume

EXAMPLES

dumpdir -f /dev/rfd0

Lists all files stored on a specific backup set of flexible disks. (Insert the first disk of that set into the disk drive before executing this command.)

dumpdir -f /dev/rfd0 | fgrep /usr/richardw/lohengrin

Checks to see if the file /usr/richardw/lohengrin is on a specific backup set. (Insert the first disk of that set into the disk drive before executing this command.)

SYNTAX
dumpdir [**-f** device]

PARAMETERS

-f device Specifies the device upon which the dump volume is mounted. If you don't specify device, **dumpdir** assumes that the device is /dev/rfd0.

EXPLANATION
The dumpdir command reads flexible disks or magnetic tapes dumped with the dump command. (An individual disk or tape is generically known as a dump volume.) The **dumpdir** command lists the names and inode numbers of all the files and directories on the dump volume(s).

FILES
rst* temporary files used by **restor**

SEE ALSO
dump
restor

echo -- echo message to terminal

EXAMPLES

```
$ echo  'Echo this message'
Echo this message
```

The message, **"Echo this message"**, is displayed on the terminal.

```
$ echo `date`
$ Tue Feb 22 14:53:35 PST 1984
```

Causes the output from the **date** command to be displayed on the terminal.

SYNTAX

echo [**-n**] [arg...]

PARAMETERS

-n Suppresses newline generation. That is, no carriage return is generated so the next material printed is appended to the same output line. This parameter **must** precede all other arguments.

arg An argument to be echoed to standard output.

EXPLANATION

The **echo** command writes its arguments (separated by blanks and terminated by a newline) to standard output.

The **echo** command is also useful for displaying diagnostic messages from shell programs, and for sending data through pipes. The following command sends "message" to the standard error when included in a shell script:

```
echo message 1>&2
```

The **"1>&2"** characters redirect standard output to standard error.

ed -- invoke TNIX text editor

SYNTAX
ed [**-**] [**-cxp**] [file]

PARAMETERS

-c Suppresses the printing of character counts by the **ed** commands **e** (edit), **r** (read), and **w** (write).

-x Causes **ed** to turn on its prompt character (*).

-p Used only with the **x** command of the optional **8560 Series Native Programming Package.** If **-x** is present, an **x** command is simulated first to handle an encrypted file.

EXPLANATION
The TNIX text editor **ed** is a line-oriented editor that allows you to create any text--program code, command files, or correspondence. Usually, you will enter the text, correct or modify it, and store it in a file for immediate or future use.

For a detailed explanation of **ed**, see **The TNIX Editor** section of the **8560 Series System Users Manual.**

ED REFERENCE SUMMARY

Table 1-5 summarizes each **ed** command. The tables in this subsection use the following notation:

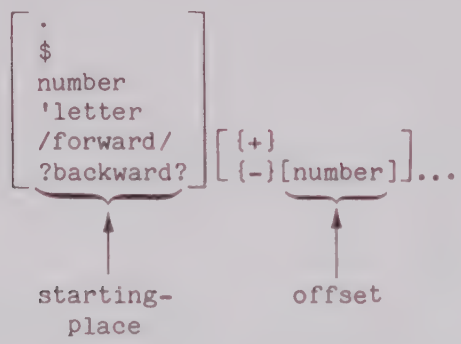
Boldface characters are entered literally.

Underline characters represent parameters for you to enter.

Brackets [ ] surround optional parts of the command.

n represents an address: a line, range of lines, search string, mark, or special address character.

Line addresses represented by n have the following formal syntax:



In the absence of a starting-place, "the current line" is assumed. In the absence of an offset, 1 is assumed. For example:

```
?here?-2
```

addresses the second line before the first occurrence of "here", searching backwards.

Table 1-6 summarizes searches and regular expressions in **ed**.

As additional reference, you will find an alphabetical list of **ed** commands in the online manual page by entering the command **man ed**.

Table 1-5
ed Command Quick Reference

Task	Command and Usage	Result
Create Text	a na text .	Appends text below line n.
	i ni text .	Inserts text above line n.
Change Text	c nc text .	Changes text of line(s) n.
	d nd	Deletes line(s) n.
	s ns/str1/str2/	Substitutes str2 for str1 in line(s) addressed by n.
	u u	Undoes substitution on last line substituted, if that line is current line.
Search	/ /string/	Searches forward for next occurrence of string in file, prints line if found.
	// //	Repeats last forward search.
	? ?string?	Searches backward for previous occurrence of string in file, prints line if found.
	?? ??	Repeats last backward search.
Move Text	m nmd	Moves (appends) line(s) addressed by n to destination line d.
	t ntd	Transfers (copies and appends) line(s) to destination line d.
	j nj	Joins addressed range of lines or, if no lines are addressed, joins current line with following line.
Print Text	l nl	Lists addressed lines.
	p np	Prints addressed lines.

Table 1-5 (Cont)
ed Command Quick Reference

Task	Command and Usage	Result
Manipulate Files	e e [file]	Edits a new file--loses old buffer contents.
	E E [file]	Edits a new file--loses old buffer contents. No warning if you haven't written (saved) old contents first.
	f f	Prints current filename.
	f f file	Changes current filename to file.
	r nr [file]	Reads (appends) file at addressed line.
	w nw [file]	Writes addressed lines to file. Default is entire contents of buffer.
	W nW [file]	Writes (appends) addressed lines to file. Default is entire contents of buffer.
Miscel-laneous	P Px	Sets prompt character to x.
	k nka	Marks addressed line with symbol a. This line now may be addressed as 'a.
	g ng/expr/cmds	Global commands: within range of lines addressed by n, executes cmds on each line containing expr. (a)
	v nv/expr/cmds	Global commands: within range of lines addressed by n, executes cmds on each line not containing expr. (a)
	x xkey	Uses key to encrypt and decrypt files when using w, r, and e commands. Used only with crypt command of Native Programming Package.
	= n=	Returns number of addressed line.
	! !command	Escapes to TNIX shell and executes command.
	!sh !sh	Escapes to a new TNIX shell and executes TNIX commands until CTRL-D is entered.
Exit from ed	q q	Exits from ed.
	Q Q	Exits from ed. No warning if you haven't written (saved) first.

(a) cmds are any ed commands except g or v; expr is any regular expression.

Table 1-6
Search and Regular Expression Quick-Reference

Task	Command	Result
Searches	/string/	Searches for first occurrence of string, beginning at ".+1".
	?string?	Searches backwards for first occurrence of string, beginning at ".-1".
	g/string/	Searches for every line containing string.
	v/string/	Searches for every line not containing string.
Regular Expressions	.	A period matches any single character.
	\	Removes the special significance of the following character (except "(", ")", or a digit).
	[chars]	Matches if the character found is in the set of chars.
	^	Matches at the beginning of the line.
	$	Matches at the end of the line.
	*	Matches a sequence of 0 or more occurrences of the preceding character.
	&	Used on the right side of a substitution command, "&" matches the string on the left side.
	\(\)	Backslash-parentheses surround an identifiable part of the search string (left side) in a substitution. On right side of the substitution, \1 represents item in first pair of backslash parentheses, \2 represents item in second pair, etc.

FILES

/tmp/e*

ed.hup Work is saved here if terminal hangs up.

SEE ALSO
See The TNIX Editor section of the 8560 Series System Users Manual.

ERROR MESSAGES

?name Inaccessible file.

? Errors in commands.

?TMP For temporary file overflow.

expr -- evaluate arguments as an expression

EXAMPLES

a=`expr $a + 1`

Adds 1 to the shell variable a.

SYNTAX
expr expression

PARAMETERS

expression An expression to be evaluated. See the following explanation.

EXPLANATION
The arguments are taken as an expression. After evaluation, the result is written to standard output. Each token of the expression is a separate argument.

The operators are listed below in order of increasing precedence.

expr1 | expr2

Yields expr1 if it is neither null nor "0", otherwise yields expr2. For example,

expr 0 '|' 1

yields **1.**

expr1 & expr2

Yields expr1 if both expr1 and expr2 are null or "0", otherwise yields "0".
For example,

expr xyz '&' abc

yields **xyz.**

expr1 relop expr2

Yields "1" if the indicated comparison is true, "FBO" if false. Here, relop is a relational operator (<, <=, =, !=, >=, or >). The comparison is performed numerically if both expressions are integers. Otherwise it is performed as an alphanumeric string comparison.

For example,

```
expr  abc '<' abd
```

yields 1 (true), while

```
expr  0 '>' 1
```

yields 0 (false).

expr1 + expr2
Adds expr1 to expr2.

expr1 - expr2
Subtracts expr2 from expr1.

expr1 * expr2
Multiplies **expr2** by **expr1**.

expr1 / expr2
Divides **expr1** by **expr2**.

expr1 % expr2
Yields the remainder of the arguments.

expr1 : expr2
Compares the string first argument with the regular expression second argument. Regular expression syntax is the same as that of the **ed** command. The \(...\) pattern symbols can be used to select a portion of the first argument. Otherwise, yields the number of characters matched ("0" on failure).
For example,

```
expr  Abcd : [Aefg]
```

yields 1 for a match with "A" in both strings.

(expr)
Parentheses for grouping.

SEE ALSO
ed
sh
test

EXIT STATUS

The **expr** command returns the following exit codes:

0	The expression is neither null nor "0".
1	The expression is null or "0".
2	The expression is invalid.

fbr -- backup and restore files

EXAMPLES

fbr -c -l 'pwd'

Archives the subtree rooted at the current directory and uses the current directory name as the archive comment.

find /usr -mtime -2 -print | fbr -c -l 'incremental' -

Archives all user files and directories that have changed in the last two days.

SYNTAX

fbr -{**cruxtd**} [**-viw**] [**-l** comment] [**-m** directory] [**-f** archive] [**-**] [file...]

PARAMETERS

To inform **fbr** what function it is to perform, one of the following function parameters (**-cruxtd**)) must be chosen.

NOTE

It is suggested that the file arguments supplied be relative to the current directory, for example "abc" rather than "/usr/dmr/abc". Files stored in this way are more easily transported between users and installations.

-c create -- Creates a new archive. The archive is initialized prior to writing the named files to it. If **-c** is specified with no file argument, "." is the default.

-r replace -- The named files are written on the archive. If files with the same names already exist on the archive, they are replaced. "Same" is determined by string comparison, so "abc" can never be the same as "/usr/dmr/abc", even if "/usr/dmr" is the current directory. If no file argument is given, "." is the default.

-u update -- Updates the archive. **-u** is similar to **-r**, but replaces a file only if it has changed since it was last archived.

-x extract -- Extracts the named files from the archive to the filesystem. The access and modification date-times and the mode are restored. If **fbr** is being run by the superuser, the owner and owner's group are also restored. If no file argument is given, the entire contents of the archive are extracted.

-t table -- Table of Contents. Lists the names of the specified files. If no file argument is given, the entire contents of the archive are listed.

-d delete -- Deletes the named files from the archive. If no file argument is given, no files are deleted.

The following parameters may be used in addition to the parameter that informs **fbr** what function it is to perform.

-v verbose -- Normally **fbr** does its work silently. The **-v** (verbose) parameter, when used with functions other than **-t**, causes **fbr** to type the name of each file it processes preceded by the function name. With the **-t** function, **-v** prints information from the archive label, as well as the following information concerning each file:

- the file's mode
- the username and groupname of its owner
- its size
- its modification date-time
- the starting block number of its data on the archive
- the name of the file

For all functions, **-v** causes an archive space usage message to be printed at the completion of the command.

-i ignore -- Errors detected while reading the archive are noted, but are ignored (no action is taken). Normally, errors cause a return to the command level. This parameter is only useful for performing table of contents (**-t**) and file extraction (**-x**) functions.

-w wait -- Causes **fbr** to wait before treating each file, to type the function name and the filename (as with **-v**), and to await the user's response. A response of **y** means "yes" and the file is processed. A response of **n** or a null response means "no" and the file does not take part in whatever is being done. Response **N** (uppercase) is the same as answering **n** to this and **all** further questions. Response **Y** (uppercase) is the same as answering **y** to this and **all** further questions. An end-of-file on standard input or an interrupt at this point will terminate **fbr** without modifying the archive or the filesystem.

This flag is ignored if a table of contents function (**-t**) is specified.

-l comment
When used in conjunction with any function that alters the archive, such as **-d**, **-r**, **-u**, or **-c**. This parameter causes **fbr** to write the comment string into the archive label if comment is not a null string. Otherwise, **-l** is ignored.

-f archive
: Uses file archive, rather than the flexible disk, as the archive medium. If this option is used with the -c function and archive already exists and has a non-zero size, that size will be the size of the created archive. Otherwise, the archive that is created will contain the specified files only and will have no free space.

-m directory
: Moves the specified files relative to directory, rather than the root or current directory, by temporarily prepending directory to each pathname to be operated on. Useful only with the -x function parameter.

EXPLANATION

The fbr command saves and restores directories and files on a flexible disk archive, preserving aliases (multiple links to the same file). Its actions are controlled by the function parameters and other optional arguments. Other arguments to the fbr command are file or directory names specifying which files are to be transferred, deleted, or listed. A parameter consisting only of a dash indicates that file and directory names should be read from standard input, one name per line. If reading names from standard input is not specified, the appearance of a directory name in the parameters refers to all files and refers recursively to subdirectories of that directory. If the function parameter is -x, -t, or -d, filenames may contain regular expressions of the form acceptable to the shell. Note that such expressions must be quoted so that the shell does not attempt to interpret them.

FILES

/tmp/fbr??????
/dev/fd0
/etc/passwd
/etc/group
/bin/mkdir or /usr/bin/mkdir

SEE ALSO

find
sh

NOTES

Pathnames are limited to 106 characters.

Multiple links to a file can only be recognized as such if all the links in question are backed up or restored in the same fbr invocation.

It is not a good practice to specify query (-w) and "read names from standard input" ("-"), because names are read until an end-of-file (EOF), and the query exits on EOF.

ERROR MESSAGES

There are several error messages. The non-obvious ones are:

Phase error The file changed between the time it was selected for archiving and the time that it was archived.

Out of memory Not enough memory is available to store all the filenames to be operated on. The number of names and/or their lengths must be reduced.

find -- find files

EXAMPLES

```
find /usr/miken \( -name a.out -o -name '*.o' \)  -exec rm {} \;
```

Removes all files named "a.out" or ending with ".o", starting from the /usr/miken directory and descending into all subdirectories.

```
find /usr/jonk -name module.l -print
```

Searches through all the directories under the /usr/jonh directory looking for the file module.l. When it is found, the complete pathname is displayed.

```
find /usr/jonk -type d -print -exec ls {} ;
```

Prints each directory (in /usr/jonk) and its contents (recursively).

SYNTAX

find pathname... expression

PARAMETERS

The following parameters are used to form the boolean **expression**. The argument n is used as a decimal integer where +n means more than n, -n means less than n, and n means exactly n.

-name filename
: True if the filename argument matches the current file name. Normal Shell argument syntax may be used if escaped (watch out for "[", "?", and "*").

-perm onum
: True if the file permission flags (bits 0777) exactly match the octal number onum (see **chmod**). If onum is prefixed by a minus sign, more flag bits (017777, see **stat** in Section 2 of this manual) become significant and the permission is tested for existence of the bits in onum rather than exact match with onum.

-type x
: True if the type of the file is x, where x is **b**, **c**, **d** or **f** for block special file, character special file, directory, or plain file, respectively.

-links n
: True if the file has n links.

-user uname
: True if the file belongs to the user uname (login name or numeric user ID).

-group gname
: True if the file belongs to group gname (group name or numeric group ID).

-size n True if the file is n blocks long (512 bytes per block).

-inum n True if the file has inode number n.

-atime n True if the file has been accessed in n days.

-mtime n True if the file has been modified in n days.

-exec command
True if the executed command returns a zero value as exit status. The end of the command must be punctuated by an escaped semicolon; that is, a semicolon must entered as "\;". A command argument "{}" is replaced by the current pathname.

-ok command
Like **-exec**, except that the generated command is written to standard output, and then standard input is read. The command will be executed **only** if you respond by entering a "y".

-print Always true; causes the current pathname to be printed.

-newer file
True if the current file has been modified more recently than the argument file.

EXPLANATION
The **find** command is used to search for files with specified characteristics and to perform various actions if the file is found. For example, **find** can print the pathnames of all files belonging to a particular user. The **find** command searches for files starting in the directory pathname and continues searching down through all subdirectories seeking files for which expression is true. Expression is boolean and is written using the parameters above. Expression also contains the actions that can be taken when the file is found, the **-exec**, **-ok**, and **-print** parameters. (These parameters also may be used as boolean tests.)

The parameters may be combined using the following operators (in order of decreasing precedence):

1) A parenthesized group of parameters and operators. Parentheses are special to the Shell and must be escaped; that is, a parentheses must me entered as "\(" or "\)".

2) The negation of a parameter ("!" is the unary not operator).

3) Concatenation of parameters (the "and" operation is implied by the juxtaposition of two parameters).

4) Alternation of parameters ("-o" is the "or" operator).

FILES
/etc/passwd
/etc/group

SEE ALSO
For information on the **filsys** file format, see **filsys** in section 5 of this manual.

For information on the **stat** system call, see **stat** in section 2 of this manual.

chmod
sh
test

format -- format a fixed or flexible disk

EXAMPLES

```
fbr filename to boot> format 0
```

In this example, the **format** command on the Standalone Utilities disk formats the internal 8560 fixed disk. **Format** destroys all previous data on the fixed disk.

```
fbr filename to boot> format 1
```

In this example, the **format** command on the Standalone Utilities disk formats the first 8503 Disk Expansion Unit attached to your system. Before booting the Standalone Utilities disk to execute **format**, perform a complete backup of the filesystem that includes the 8503. **Format** destroys all previous data on the 8503.

```
# format
```

In this example, **format** formats a double-density flexible disk. The system must be in single-user or multi-user mode.

```
# format -s
```

In this example, **format** formats a single-density flexible disk. The system must be in single-user or multi-user mode.

SYNTAX

format [drive] [Standalone Utilities version, for fixed disks]

format [**-s**] [standard TNIX version, for flexible disks]

PARAMETERS

-s Formats a **single**-density flexible disk. (The default is to format **double**-density flexible disks.)

drive Formats the specified fixed disk drive. The values for drive may be 0, 1, 2, or 3 (logical devices /dev/rhd0, /dev/rhd1, /dev/rhd2, and /dev/rhd3, respectively).

EXPLANATION

If the contents of a fixed disk drive become corrupted, it may be necessary to reformat the disk. You should try correcting the problem with **syschk** first, since **syschk** can solve most problems of this type. If **syschk** cannot repair the disk, the Standalone Utilities version of **format** redefines various logical fields in each fixed disk sector (such as ID fields and data fields). After you reformat the disk, use the **mkfs** command to create a new filesystem on the disk and run **syschk** to verify the filesystem's integrity. Once a usable filesystem exists on the disk, you can restore files with the **restor** command. Be sure to reinstall the boot block (with the **mkboot** command) on the disk as well.

CAUTION

Formatting a disk **destroys** any previous data on the disk.

The standard TNIX **format** command (used for flexible disks) functions analogously to the Standalone Utilities version of **format**. Always format flexible disks before attempting to use them. (Be sure that the write-protect slot is covered by a piece of tape or similar opaque material.)

SEE ALSO

mkboot
mkfs
restor
syschk

FILES

/dev/rfd0 The 8560 flexible disk drive's name.

/dev/rhd* Names for fixed disks within the 8560 and 8503 DEUs.

getty -- read tty characteristics

SYNTAX
/etc/getty [char]

EXPLANATION
The **getty** command is automatically invoked by **init** immediately after a terminal comes online. **Getty** reads the user's login name and automatically calls the **login** command with the name as argument. While reading the name, **getty** attempts to set TNIX to the baud rate and the type of terminal being used.

Init calls **getty** with a single character argument: the second character in the line for the terminal found in the ttys file. This argument determines a sequence of line baud rates through which **getty** cycles.

If the terminal's "break" key is depressed, **getty** cycles to the next speed appropriate to the type of line and prints the greeting message again.

The following arguments from the **ttys** file are:

0 Cycles through 9600-4800-2400-1200-600-300 baud. Useful as a default for a variety of on-line terminals.

1 Same as "0" but starting at 4800 baud.

2 Same as "0" but starting at 2400 baud.

3 Same as "0" but starting at 1200 baud.

4 Same as "0" but starting at 600 baud.

5 Same as "0" but starting at 300 baud.

6 Starts at 1200 baud, cycles to 300 and back. Useful with 212 datasets where most terminals run at 1200 baud.

7 Same as "6" but starts at 300 baud.

A Intended for on-line CRT terminals such as TEK 4105 at 9600 baud.

B Same as **A** except 4800 baud.

C Same as **A** except 2400 baud.

D Same as **A** except 1200 baud.

E Same as **A** except 600 baud.

F Same as **A** except 300 baud.

a-f Same as **A-F** except HSI protocol will be forced online even if jumped for RS-232.

SEE ALSO

For more information on the **ioctl** system call, see **ioctl** in Section 2 of this manual.

For more information on the **ttys** file format, see **ttys** in Section 5 of this manual.

init
login

grep, egrep, fgrep -- search a file for a pattern

EXAMPLES

fgrep March amber	Prints any line that contains the word March in file amber.
grep '^.XX' letter	Prints all lines in the letter file that begin with ".XX".

SYNTAX

```
      [-c]
      [-l]
grep  [-s] [-vnbhy] [[-e] expression] [infile ...]

      [-c]
      [-l]
egrep [-s] [-vnbh] [-f file] [[-e] expression] [infile ...]

      [-c]
      [-l]
fgrep [-s] [-vnbhx] [-f file] [string] [infile ...]
```

PARAMETERS

-c Only a count of matching lines is printed.

-l The names of files with matching lines are listed, separated by newlines.

-s No output is produced, but the exit status is generated.

-v All lines except those matching are printed.

-n Each line is preceded by its line number in the file.

-b Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.

-h Causes all output lines to be printed without the filename header.

-y Lowercase letters in the pattern also match uppercase letters in the input (**grep** only). This is particularly useful when you don't know the case (upper or lower) of certain letters in the pattern.

-x Exact -- only lines matched in their entirety are printed (**fgrep** only).

expression
A limited regular expression (**grep**) or an extended regular expression (**egrep**).

-e expression
Same as expression, but used only when the first character of expression is a dash (-).

infile The pathname of the input file to be searched for a particular pattern.

-f file The regular expression (**egrep**) or string list (**fgrep**) is taken from file rather than from the command line.

string A fixed (literal) string of characters.

EXPLANATION

Commands of the **grep** family search the input files (standard input default) for lines matching a pattern. Normally, each line containing the matching pattern is copied to standard output. If there is more than one input file specified, the filename precedes each output line, unless overridden by the **-h** parameter.

There are three pattern recognition commands: **grep**, **egrep**, and **fgrep**. Their main differences are as follows:

grep The patterns are limited regular expressions in the style of **ed**; it uses a compact nondeterministic algorithm.

egrep The patterns are extended regular expressions; it uses a fast deterministic algorithm that sometimes needs exponential space.

fgrep The patterns are fixed (literal) strings; it is fast and compact.

Care should be taken when using the characters **$ * [^ | ? ' " ( )** and **** in expression as they are also meaningful to the Shell. It is safest to enclose the entire expression argument in single quotes ('expression').

The **fgrep** command searches for lines that contain one of the (newline-separated) strings.

The **egrep** command accepts extended regular expressions. In the following description, "character" excludes newline:

A backslash (\) followed by a single character matches that character.

The caret (^) matches the beginning of a line.

The dollar sign ($) matches the end of a line.

A period (.) matches any character; that is, a wildcard character.

A single character not otherwise endowed with some special meaning matches that character.

A string enclosed in brackets [ ] matches any **single** character from the string. For instance, the pattern **'section[149]'** would match with any line containing the words **section1**, **section4**, or **section9**. Ranges of ASCII character codes may be abbreviated as in "a-z", or "0-9". For example, the pattern **'section[1-3]'** would match with any line containing the words **section1**, **section2**, or **section3**. A right bracket (]) may occur only as the first character of the string. A literal dash (-) must be placed where it can't be mistaken as a range indicator.

A regular expression followed by * (+, ?) matches a sequence of 0 or more (1 or more, 0 or 1) matches of the regular expression.

Two concatenated regular expressions match a match of the first followed by a match of the second.

Two regular expressions separated by | or newline match either a match for the first or a match for the second.

A regular expression enclosed in parentheses matches a match for the regular expression.

The order of precedence (from highest to lowest) of operators at the same parenthesis level is:

- [ ]
- * + ?
- concatenation
- |
- newline

SEE ALSO
ed
sed
sh

NOTES
Lines are limited to 256 characters; longer lines are truncated.

EXIT STATUS

The **grep**, **fgrep**, and **egrep** commands return the following exit codes:

0	One or more matches have been found.
1	No matches were found.
2	One or more syntax errors occurred, or a file was inaccessible.

help -- a guide to online documentation

SYNTAX
help

EXPLANATION
When you type **help**, the system displays the following message:

For online information about your 8560 Series system and its TNIX operating system, use the "man" and "index" commands.

Use "man" if you know the name of the command you want:

```
$ man make          describes the "make" command
$ man map           describes the 8540's "map" command
$ man index         gives more information about the "index" command
```

Use "index" to find out what information is available on a given topic, for example:

```
$ index run program
$ index delete file
$ index Pascal
```

For more information about your 8560 Series system and TNIX, see the 8560 Series System Users Manual.

SEE ALSO
index
man

index -- keyword index to online manual pages

EXAMPLES

index directory	Lists online manual pages that have information about directories.
index delete directory	Lists online manual pages that have information about deleting directories.

SYNTAX

index [keyword...]

PARAMETERS

keyword A word that describes the topic for which you need help. The word may contain any non-blank character. Long words are truncated to 14 characters. Capitalization is ignored. Multiple keywords may be entered in any order.

EXPLANATION

The **index** command lists every on line manual page that describes the keyword (or group of keywords) that you specify.

The **index** command recognizes certain groups of synonyms. For example, the following commands are equivalent:

```
index remove
index delete
index erase
```

By convention, keywords are stored in their simplest forms. Thus the **index** command does not recognize variations on keywords. For example, use **"delete file"**, not **"deleting files"**.

SEE ALSO

help
man

init -- initialize system processes

SYNTAX
/etc/init

EXPLANATION
The **init** command is invoked as the last step of the boot procedure. When invoked, **init** creates a process for each terminal on which a user may log in.

First, **init** invokes a shell that executes the commands in the file **/etc/rc**. The commands in /etc/rc are used to perform housekeeping tasks like removing temporary files, mounting file systems, and starting system daemons.

Then **init** reads the file /etc/ttys and creates a process for each terminal specified in the file. Each of these processes opens the appropriate terminal for reading and writing.

Opening the terminal may involve a delay, since the **open** system call is not completed until a functioning terminal is connected to a port on the 8560. When a terminal is connected, **init** automatically calls the **/etc/getty** command with the second character of the line for the terminal in the ttys file, as the argument. The **getty** command reads the user's name and invokes the **login** command to log the user in and execute the shell.

When the **logout** command is used, the user's shell terminates. The **init** command waits for the **logout** and removes the appropriate entry from the file utmp, which records current users. Then the appropriate terminal is reopened and **getty** is reinvoked.

The **init** command has some additional functions:

- The **init** command catches the hangup signal SIGHUP and interprets it to mean that the system should be brought from multi-user to single-user. To bring the system from multi-user to single-user, enter the command: **kill -1 1**. (You must be logged into the "root" account.)

- The init command also catches the interrupt signal SIGINT. This signal means that **init** is to re-read /etc/ttys and close any terminals that have been disabled. It will also open any new terminals that have just been connected. This allows you to reset your terminal configuration without rebooting the system. To do this enter: **kill -2 1**. (You must be logged into the "root" account.)

FILES
/dev/tty?
/etc/utmp
/etc/ttys
/etc/rc

SEE ALSO

For information on the **ttys** file format, see **ttys** in Section 5 of this manual.

getty
kill
login
sh

install -- install software

SYNTAX
install

EXPLANATION
The **install** command installs new or updated software supplied on a TEKTRONIX Software Distribution Diskette on your 8560. Here's how to install Tektronix-supplied software on your system:

1. Log into the "root" account on the system console.

2. Make sure that you are the only person logged into your system.

3. Perform a full backup of your system. For information on performing a full backup of your system, see **dump**.

4. Insert the Software Distribution Diskette containing the new software into the flexible disk drive and close the drive's door.

5. Install the new software by typing:

   # **install**

   When the "#" prompt is displayed, the new software is installed.

6. Remove the Software Distribution Diskette from the flexible disk drive.

7. Perform an incremental level 1 backup of your system. This backup records all changes made to your system after the full backup that you performed in step 1 of this procedure. See the discussion of the **dump** command for information on performing an incremental, level 1 backup of your system.

8. Reboot your system, then resume normal operations. Type the **shutdown** command, wait for **shutdown** to display the message telling you that it is safe to reboot your system, then toggle the RESTART switch.

The new software is now installed on your system.

kill -- terminate a process

EXAMPLES

kill 17196 Kills the process with the Process IDentification number (PID) of **17196**. Even though this is the most-used form, it will not kill all processes.

kill -9 16334 Kills the process with the Process IDentification number (PID) of **16334**. This form kills **any** process specified (that you started), regardless of what signal(s) the process may trap.

SYNTAX
kill [-signal] processid...

PARAMETERS

signal Specifies what kind of signal to send to the specified process. The allowed range is 0--25. Type **man signal** and **man sigsys** for a definition of each signal.

processid The Process IDentification number (PID). If you don't know the PID number of the process, use the **ps** command to find out.

EXPLANATION
The **kill** command sends the value of signal to the specified process(es). If signal is not specified, then signal 15 (software termination signal) is sent to the process(es). Some processes may catch, hold, or simply ignore the signal. In such cases, you can use signal 9 (kill process), which cannot be caught, held, or ignored.

By convention, if you specify process number 0, **all** processes started by your login are terminated and you will be logged off the system.

You cannot kill processes that do not belong to you unless you are a superuser.

One way to place a process into background is to append the "**&**" character to the command line. When this is done, the **PID** number is output to your terminal. Process numbers can also be found by using the **ps** command.

SEE ALSO
For information on the **kill** system call, see **kill** in Section 2 of this manual.

For information on the **signal** system call, see **signal** in Section 2 of this manual.

ps

ksh -- Invoke Keyshell

EXAMPLES

ksh -c Invokes **ksh** in compact mode.

SYNTAX

ksh [-C][-F][-T]
[-c][-f][-t]

PARAMETERS

-c Compact mode. Deletes two extra lines usually included in the on-screen label area, thus providing more room for other information. Some terminals may ignore this parameter.

-C This parameter reverses the effect of including **-c** in the environment variable KSH. That is, it prevents **ksh** from running in compact mode. Recognized only in the **ksh** command line.

-f Function key numbers. Includes key numbers in the on-screen function key labels. This is the default condition.

-F This parameter reverses the effect of including **-f** in the environment variable KSH; that is, it prevents **ksh** from including numeric key labels. Recognized only in the **ksh** command line.

-t Type-ahead. Turns off type-ahead while **ksh** is running. This means that anything you type while a command is executing will be disregarded, instead of saved and acted upon when the command finishes. This is the default condition.

-T This parameter reverses the effect of including **-t** in the KSH environment variable; that is, it causes type-ahead to remain in effect while **ksh** is running. Recognized only in the **ksh** command line.

EXPLANATION

The **ksh** program provides the Keyshell interface (Keyshell is described in the **8560 Series System Users Manual**). Ordinarily, the .profile command file automatically invokes **ksh** whenever you log in. You will seldom need to invoke **ksh** yourself.

The .profile command file also sets the **ksh** environment variable KSH. The value of KSH determines the default behavior of **ksh**. Suppose, for example, that KSH contained the **-f** parameter. Then invoking **ksh** without parameters or arguments would cause it to act as if invoked with:

$ ksh -f

By using the uppercase command parameters, you can reverse the effect of the KSH variable. For example, this command line:

```
$ ksh -F
```

reverses the effect of using **-f** in the KSH environment variable.

LEAVING KSH

Ordinarily, you will leave **ksh** by pressing the exit function key on the Keyshell top level. However, you can also return to unassisted communication with the shell by typing:

```
$ exit <CR>
```

Exit terminates **ksh** and leaves the Keyshell function keys inactive. **Exit** has no effect if **ksh** is not running.

SEE ALSO
The Keyshell section in the 8560 **Series System Users Manual.**

The 8560 System Managers Guide

NOTES
Other **ksh** parameters are for Tektronix use only and are not described here. Do not use any parameters other than those described here, and do not change any other parameters you find in your .profile command file.

The System Manager must have set up your account with **ksh** variables. Otherwise the following error message will be displayed:

```
ksh: No session file
```

ldgpib -- initialize the GPIB Interface

SYNTAX
/etc/ldgpib [filename]

PARAMETERS
filename Specifies the file that contains the GPIB Interface monitor code. If you don't specify filename, **ldgpib** automatically uses /etc/gpib/code.

EXPLANATION
The **ldgpib** command initializes the optional GPIB Interface. The command loads the GPIB Interface monitor code into the GPIB Interface, and establishes communication mechanisms between the GPIB Interface and the 8560. The **ldgpib** command then directly executes **mkgpib**, which re-establishes the GPIB device definition list within the GPIB Interface.

If your system contains a GPIB Interface, the /etc/rc file should include **/etc/ldgpib**. This way, the GPIB Interface is initialized every time you boot TNIX.

NOTE

The GPIB Interface board must be installed before you execute this command.

FILES

/etc/gpib/code The standard GPIB Interface monitor code.

/etc/gpib/dev Contains the definitions of all GPIB-compatible devices currently associated with the system.

/dev/gpib The 8560 GPIB Interface device.

SEE ALSO
For further GPIB-related information, see the 8560 **Magnetic Tape Interface Users Manual**.

For information on the **gpib** file, see **gpib** in Section 4 of this manual.

For information on the **gpib** file format, see **gpib** in Section 5 of this manual.

mkgpib

NOTES

You must be logged in as root to execute **ldgpib**.

Don't try to execute **ldgpib** if your system doesn't have a GPIB Interface installed. **If you do, the system may** crash.

If you remove the GPIB Interface from your system, be sure to delete **ldgpib** from the /etc/rc file, and to remove /dev/gpib.

If you execute **ldgpib** any time after system initialization, any GPIB operations that are in progress will be disrupted.

Use any monitor code other than that contained in /etc/gpib/code at your own risk.

ERROR MESSAGES

cannot initialize GPIB system

Indicates that the GPIB Interface couldn't load or execute its monitor code. Either the monitor code is bad, or the GPIB Interface has malfunctioned. Try executing **/etc/ldgpib** again. If the problem persists, contact your Tektronix field service representative.

cannot initialize GPIB queuing

Indicates that either the GPIB Interface or the system processor has malfunctioned. If the problem persists, contact your Tektronix field service representative.

Since **ldgpib** invokes **mkgpib**, any message generated by **mkgpib** may also be displayed.

The remaining error messages are self-explanatory.

libgen -- invoke library generator

EXAMPLES

libgen -n newlib -i fpadd fpsub fpmult

Causes the object modules from files fpadd, fpsub, and fpmult to be inserted into a new library named newlib.

libgen -o mylib -n newlib -d mymod oldmod

Causes the library modules from the old library (mylib) to be copied into the new library (newlib), with the exception of library modules mymod and oldmod.

libgen -o mylib -n newlib -i io.obj

Causes the library modules from the old library (mylib) to be copied into the new library (newlib), along with the object module from file io.obj.

SYNTAX

libgen [**-c** commandfile] [**-d** modulename...] [**-h** string]
[**-i** objfile...] [**-n** newlib] [**-o** oldlib] [**-r** objfile...]
[**-v**] [**-x** modulename [pathname]]

PARAMETERS

-c commandfile
Invokes **libgen** command file commandfile. A command file contains a series of **libgen** parameters to process.

-d modulename...
Deletes the specified object module(s) from the new library.

-h string
Specifies the identification header for the new library. The string may contain any printable ASCII characters but should not start with a dash (-). The maximum length of string is 80 characters.

-i objfile...
Inserts the object module from file objfile into the new library. More than one object file may be specified.

-n newlib
Specifies the pathname of the output file that receives the updated (**new**) library.

-o oldlib
Specifies the pathname of the input file that contains the old (**existing**) library.

-r objfile...
Replaces an old library module with a new object module from file objfile when generating a new library. The library module name should have the same name as the new object module within file objfile. Otherwise, the new object module is inserted into the library. More than one object file may be specified.

-v Specifies a verbose listing. Outputs the **libgen** banner (which contains the version number) and each **libgen** command as it is processed.

-x modulename [pathname]
Extracts (copies) object module modulename from the old library file oldfile into file pathname.

EXPLANATION

The libgen (**lib**rary **gen**erator) command invokes a general-purpose utility program used to create and maintain object module libraries for use with the linker.

Using libgen, you can collect assembler-generated or compiler-generated object modules into library files. From these library files, the object modules can be individually accessed by the linker, based on the information provided by each object module.

FILES

###.lib.tmp.nn A temporary file (where **nn** is an integer).

SEE ALSO

8500 Modular MDL Series Assembler Core Users Manual for B Series Assemblers

lstr

NOTES

The name of an object module stored in an object file is independent of the object file's name. For instance, object file calc.obj could contain an object module with the name subsmod3. The module name (subsmod3 in this case) is assigned by the **NAME** assembler directive.

link -- link object modules

EXAMPLES

link -v -o loadfile -O objfile1 objfile2

This single-line linker invocation causes the linker to:

- output each linker command as it is processed, along with the statistics listing.
- link the object modules from files objfile1 and objfile2
- generate the load module file (loadfile) containing the linked object code

link -v -c commandfile1

This command-file linker invocation causes the linker to:

- output each linker command as it is processed
- output the statistics listing to the system terminal
- execute all commands in command file commandfile1
- **not** produce a load file (because the **-o** parameter is not specified)

SYNTAX
link [**-CDLOcdmorstvx**] [parameters]

PARAMETERS
Each of the following parameters are also known as linker command options.

-C Assigns a classname to one or more section(s).

-D Defines a global symbol at link time.

-L Locates a class or section to a specified memory area.

-O Specifies the object modules to be linked. Object modules may be from the following types of files:

- files generated by an assembler or a compiler
- relinkable load files generated by the linker with the **-r** parameter
- library files generated by **libgen**

-c Invokes a linker command file.

-d Directs the linker to store debug information in the load module.

-m Defines a memory map configuration.

-o Specifies the load module filename. The linker writes the linked object code into this file.

-r Directs the linker to store relink and debug information in the load module.

-s Specifies which linked load files (previously generated by the linker using either the **-d** or **-r** parameter) are to be linked. **Only** global symbol values are extracted from these files.

-t Defines the relocation type (ALIGN, BYTE, INPAGE, or PAGE) of the specified sections.

-v Specifies a verbose listing. Outputs the linker banner with the version number, each linker command option as it is processed, and the statistics listing.

-x Specifies the load module transfer address.

EXPLANATION

The link command merges one or more independently assembled object files into a load file suitable for loading into memory. Assembler or compiler generated files, library files, or linked files may be used as linker input.

SEE ALSO

8500 Modular MDL Series Assembler Core Users Manual for B Series Assemblers

lstr

ln -- create a link to a file

EXAMPLES

ln craw.c temp.link

Creates a new link to file craw.c. The new link will be a directory entry (filename) named temp.link.

ln /usr/johnb/testfile

Creates a link to file /usr/johnb/testfile. The new directory entry will be called testfile and will be placed in your **current** directory. (The current directory cannot be /usr/johnb.)

SYNTAX
ln name1 [name2]

PARAMETERS

name1 The pathname of a file.

name2 The new pathname that you want to reference file name1 by.

EXPLANATION
The **ln** command is used to make an additional pathname for an existing file.

Each directory entry (filename) represents a file. The connection between the **actual** physical file and the directory entry used to refer to that file is called a **link.**

Using the **ln** command you may create as many links to the same file as you need. Thus more than one pathname can be used to refer to the same file.

Although a new link to a file has a new pathname, a new copy of a file has not been created. Only a new pathname is created that points to the same physical file.

Unlike copying a file, a new link does not use additional disk space for storage.

In the command syntax, name1 is one of the current pathnames for a file. With the new link formed, file name1 may be referred to by pathname name2. If name2 is not specified, the new directory entry is made in your **current** directory. The directory entry is obtained by extracting the filename portion of pathname name1.

SEE ALSO
rm

NOTES

If several users are linked to a file, a change to the file made by any user changes the file for **all** the users.

A file with several links is not removed until all the links are removed.

You cannot link to a directory or link across file systems. Only files may be linked.

login -- sign on

SYNTAX
login [username]

PARAMETERS

username An existing login name valid for your system.

EXPLANATION
The **login** command is used when you initially sign on. You may also use **login** at any time to change from one user to another. The latter case is the one described here.

If **login** is invoked without an argument, it asks for your user name, and, if appropriate, a password.

After a successful login, you are informed of any mail, and the message-of-the-day is displayed. The **login** command initializes the user and group IDs, and the working directory. Then **login** executes a command interpreter (usually **sh**) according to specifications found in the password file.

If you attempt to log on and are unsuccessful three consecutive times, the system will display a help message for you.

FILES

/etc/motd Message-of-the-day.

/etc/passwd The password file.

SEE ALSO
8560 System Managers Guide

For information on the **passwd** file format, see **passwd** in Section 5 of this manual.

getty
init
mail
newgrp
passwd

NOTES

You must have an account on the system. See the System Manager.

ERROR MESSAGES

Login incorrect Issued if the user login name or the password is bad.

No Shell The login shell specified in the **/etc/passwd** file does not exist.

lp1d, lp2d -- control line printer spooling

SYNTAX

```
        [lp1d]
/usr/lib/[lp2d]
```

EXPLANATION

Each **lp?d** is the daemon for a line printer, responsible for printing files spooled to that printer. The daemon is started automatically when a file is to be printed using the **lp1r** (**lp2r**) command. The **lp1d** (or **lp2d**) daemon uses the directory /usr/spool/lp1 (lp2) to spool the files from the **lp1r** (**lp2r**) command. The file **lock** in the /usr/spool/lp1? directory is used to prevent two daemons from becoming active simultaneously.

After the **lp?r** command has successfully placed the lock file, it starts the daemon in the background and exits.

The daemon scans the /usr/spool/lp1? directory for files beginning with **df**. Each such file was submitted as a job by the command **(lp?r)**. Each line of these job files begins with a key character to specify what to do with the remainder of the line.

L — is followed by the decimal user ID and group ID of the user who spooled this job. This line causes this job's banner page to be printed.

B — specifies that the rest of the line is the name of a file to be printed.

F — is the same as **B** except a form feed is appended to the beginning of the file.

U — specifies that the rest of the line is a file name. After the job has been transmitted, the file is unlinked.

M — is followed by a user ID. After the job is sent, a message is **mailed** to the user to verify that the job was sent.

Any error encountered causes the daemon to wait and start over. This means that an improperly constructed **df** file may cause the same job to be submitted repeatedly.

Each daemon sends its output to the corresponding /dev/lp? file. To provide a flexible line printer interface, the /dev/lp? files are links to either a /dev/aux "line printer" port or a /dev/tty "terminal" port.

NOTES

Perform the following procedure as the superuser to move the line printer link from an aux device to a tty:

1. Choose what /dev/tty?? is going to be the new line printer. For example, choose /dev/tty03 as the new /dev/lp1.

2. Change the entry for the terminal in the file /etc/ttys so that there is no login for the terminal tty03. See **ttys** in Section 5 of this manual.

3. Change the link to the new /dev/tty?? from the old /dev/aux. For example:

```
rm /dev/lp1
ln /dev/tty03 /dev/lp1
```

4. Move your line printer connector from the "line printer 1" port to the "HSI I/O 3" port.

To move the line printer back to the /dev/aux port, reverse the above process.

To restart a daemon (in the case of hardware or software malfunction), kill the old daemon if still running and remove the lock file before initiating the new daemon. This is done automatically by /etc/rc when the system is brought up in case any jobs were left in the spooling directory when the system went down.

FILES

/usr/spool/lp?	spool area for a line printer daemon
/etc/passwd	to get information required to print the banner page
/dev/lp1 /dev/lp2	line printer devices

SEE ALSO

For information on the **banproto** file format, see **banproto** in Section 5 of this manual.

For information on the **ttys** file format, see **ttys** in Section 5 of this manual.

lpr
slp
stty

lpr, lp1r, lp2r -- send a file to the line printer

EXAMPLES

lp1r file.o

Causes file file.o to print out on line printer 1.

SYNTAX
lp1r [**-cmr**] [file...]

lp2r [**-cmr**] [file...]

PARAMETERS

-c Copy the file to insulate against changes that may happen before printing.

-m Report by **mail** when printing is complete. This is automatic if **lpr** can't link to the file to print.

-r Remove the file when it has been queued.

file The pathname of a file to printed on the specified line printer.

EXPLANATION
The **lpr** command causes file to queue for printing on the default lineprinter (lineprinter 1). The **lp1r** or **lp2r** commands cause file to be queue for printing on line printer 1 or 2, respectively. If no files are named, or if the file is specified by "-", standard input is read.

FILES

/usr/spool/lp?/lock

/usr/spool/lp?/cf* The data file.

/usr/spool/lp?/df* The daemon control file.

/usr/spool/lp?/tf* The temporary version of control file.

/dev/lp?

SEE ALSO
lpd

ls, ll, lf, lr, lx -- list contents of directory

EXAMPLES

ls -l	Lists the files in the current directory, giving their sizes and other attributes.
ls -u /usr	Lists the files in directory /usr, sorted by time of last access.

SYNTAX

```
{ls}
{ll}
{lf}
{lr}                          [-u]
{lx} [-ltasdrigAfnm1CqbxFR] [-c] [file...] [directory...]
```

PARAMETERS

-l long -- Lists in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. If the file is a special file, the field size contains the major and minor device numbers instead.

-t time -- Sorts by time modified (most recent first) instead of alphabetically by filename.

-a all -- Lists all entries, including those entries whose names begin with a period (".").

-s size -- Gives size in blocks, including indirect blocks, for each entry.

-d directory -- If argument is a directory, lists only its name and not its contents. This is used mainly with -l to get a directory's status.

-r reverse -- Reverses the order of sort to get reverse alphabetic or oldest first, as appropriate.

-i i-number -- Prints the i-number in the first column of the report for each file listed.

-g group-id -- Prints the group ID instead of the owner ID in a long (-l) listing.

-u Uses time of last access instead of last modification for sorting (-t) or printing (-l).

-c Uses time of last modification to inode (mode, etc.), instead of last modification to file, for sorting (-t) or printing (-l).

-A List all entries except "." and "..". This option is normally off for users, but on for superusers. The **-A** parameter toggles the option.

-f force -- Forces each argument to be interpreted as a directory and lists the name found in each slot. This parameter turns off the **-l**, **-t**, **-s**, and **-r** parameters and turns on **-a**. The order in which entries are listed is the order in which they appear in the directory.

-n numeric ID -- uses decimal user and/or group IDs. Saves time by not accessing the /etc/passwd and/or /etc/group files for user and/or group names.

-m Forces output in stream format.

-1 one -- Outputs one entry per line.

-C Columnar -- Generates multi-column output.

-q Replaces non-displayed characters (0--37 and 177 octal) in file names with "?" before printing the filename.

-b Replaces non-displayed characters (0--37 and 177 octal) in file names with the character's octal value (displayed in the \NNN format) before printing the filename.

-x Sorts columnar output by line instead of by column.

-F Prints directory names with a trailing "/" character and executable files with a trailing "*" character.

-R Recursive -- Lists each subdirectory recursively.

file The pathname of a file.

directory The pathname of a directory.

EXPLANATION

The **ls** command lists the name of each directory and file that you specify, the contents of directory, and any other information you requested. The output is sorted alphabetically by column. The current directory is listed if you don't specify a file or directory to list.

The **ll**, **lf**, **lr**, and **lx** commands are variations on the **ls** command, as shown in the following list:

Type	To Execute
ll	ls -l
lf	ls -F
lr	ls -R
lx	ls -x

The mode printed under the **-l** parameter contains 10 characters, as shown in this example:

```
$ ls -l myfile

---------- myfile 999 kjasdlkf
```

Each character of the mode is interpreted as follows:

The **first character** is

d if the entry is a directory
b if the entry is a block-type special file, such as a disk drive
c if the entry is a character-type special file, such as a terminal
- if the entry is an ordinary file

The next nine characters are interpreted as three sets of three bits each. The first set refers to **owner** permissions; the next set to **group** permissions (those in your user group); and the last to **other** permissions (those outside of your user group). Within each set, the three characters indicate permissions respectively to **read**, to **write**, or to **execute** the file as a program. For a directory, "execute" permission is interpreted to mean permission to search the directory for a specified file. The permissions are indicated as follows:

r if the file is readable
w if the file is writable
x if the file is executable
- if the indicated permission is not granted

The group-execute permission character (**x**) is replaced with an **s** if the file has set-group-ID mode; likewise the user-execute permission character (**x**) is replaced with an **s** if the file has set-user-ID mode.

The **last character** of the mode (normally "x" or "-") is replaced with a **t** if the 1000 bit of the mode is on. See **chmod** for the meaning of this mode.

As an example, suppose the mode were listed as:

```
-rwxr-xr--
```

This would mean that the file is:

- ordinary
- readable, writable, and executable by the owner
- readable and executable by the group
- readable by others

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

NOTES

The **ls** command assumes that the newline and tab characters in file names are printing characters.

The output device is assumed to be 80 columns wide.

FILES

/etc/passwd Used to get user IDs for "**ls -l**".

/etc/group Used to get group IDs for "**ls -g**".

SEE ALSO

chmod

lstr -- list symbols found in B series object format file

SYNTAX
lstr [**-ghnosuv**] [file...]

PARAMETERS
-g Prints only global (external) symbols.

-h Prints the header which was inserted in a library with the **libgen -h** parameter.

-n Sorts numerically rather than alphabetically.

-o Prepends a file or library element name to the beginning of each output line, rather than to one line only.

-s Appends the length of the section to each section's output lines.

-u Prints only undefined symbols.

-v Prints the version number of **lstr**.

file The pathname of a file containing an object module.

EXPLANATION
The lstr command prints the symbol table of each object module in the argument list. If an argument is a library, a listing for each object file in the library is produced.

Each symbol is preceded by its value (zeros if undefined), and a single letter. The single letter must be one of the following:

- U -- undefined
- A -- absolute section
- S -- SECTION section
- D -- global data, entry, or constant
- C -- COMMON section
- R -- RESERVE section
- u -- undefined local data, entry, or constant
- l -- local data, entry, or constant

The output is sorted alphabetically.

SEE ALSO
libgen
link

mail -- send or receive mail among users

EXAMPLES

mail Displays the most recent letter and queries you for your next action.

mail -r Displays the oldest letter and queries you for your next action.

mail -p Displays all letters in your mailbox without any queries.

SYNTAX

mail person...

or

mail [-rqp] [-f file]

PARAMETERS

person The user name of the person to whome you want to send mail.

-r Causes the letters in your mailbox to be displayed in oldest to newest order. The default is to display in the order of newest to oldest.

-p Prints all the letters in your mailbox with no queries.

-q Causes **mail** to terminate when interrupted (by entering a <CTRL-C>). The mailbox will be unchanged, regardless of any deletions or changes.

-f file Causes the **mail** command to read letters from file, rather than from your mailbox file.

file The pathname of a file containing letters.

EXPLANATION

The **mail** command is used to send mail to and receive mail from other users.

Examining Your Mail

When another user mails you a letter, the letter is appended to the contents of your mailbox file. When you log in or if you are already logged in, TNIX sends you the following message (on standard output):

```
$ You have mail.
```

When you see this message, you may examine your mail by typing:

```
$ mail
```

The **mail** command then displays the latest letter and queries you for your next command. You may respond in any of the following ways:

<CR> Display the next letter by pressing the Carriage Return key.

d Delete the displayed letter from your mailbox and display the next one.

p Display the same (current) letter again.

- Display the letter immediately preceding the one currently displayed.

s [file ...]
Save the current letter by appending it to the contents of the named file. (If file is not specified, mbox is the default file.) This also deletes the message from your mailbox file.

w [file ...]
Save the current message, without a header, by appending it to the contents of the named file. (If file is not specified, mbox is the default file.) This also deletes the message from your mailbox file.

m [person ...]
Mail the letter currently displayed to the person(s) named. If you don't specify person, the message is sent to you. The next letter is then displayed.

q or <CTRL-D>
quit. Causes **mail** to terminate. A **<CTRL-D>** is generated by holding the **CTRL** key down and then pressing the **D** key once.

x Exit. Causes **mail** to stop and leaves the mailbox file completely unchanged, regardless of any deletions or modifications.

! command
Escape to the shell to perform command.

? Prints a short summary of the possible responses.

If **mail** is displaying a letter and you enter a **<CTRL-C>** (an interrupt), **mail** stops and queries you for your next action unless you included the **-q** parameter when you invoked **mail**.

How to Send a Letter
To compose a letter and then send it to another user, type:

```
$ mail person...
```

where person is the user name of the person to whom you want to send mail to. When this form of the **mail** command is specified, **mail** takes standard input until you enter a **<CTRL-D>**, or a line with just a **period**. Then **mail** appends your letter to the contents of each person's mailbox file. The

message is preceded by your login name and a postmark.

Each user owns his own mailbox file, which is by default generally readable but not writable. If desired, you may make your mailbox file unreadable.

FILES

/usr/spool/mail/*	Mailbox files.
/etc/passwd	To identify sender and locate persons.
mbox	Saved mail.
/tmp/ma*	Temporary file.
dead.letter	Unmailable text.

SEE ALSO
write

make -- maintain files

SYNTAX
make [-iknrtrs] [-f makefile][file...]

PARAMETERS

-f makefile
the given file is used as the script rather than "makefile" or "Makefile". If makefile is "-" the standard input is used as the script. More than one -f parameter may appear.

-i Equivalent to the special entry ".IGNORE:".

-k When a command returns nonzero status, abandon work on the current entry, but continue on branches that do not depend on the current entry.

-n Trace and print, but do not execute the commands needed to update the targets.

-t Touch, or update the modified date of targets without executing any commands.

-r Equivalent to an initial special entry ".SUFFIXES:" with no list.

-s Equivalent to the special entry ".SILENT:".

file The pathname of a file.

EXPLANATION
The **make** command executes commands in makefile to update one or more target files. The default is to use the script in "makefile" (or "Makefile" if "makefile" does not exist.)

The **make** command updates or creates a target if it depends on prerequisite files that have been modified since the target was last modified, or if the target does not exist.

Makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated list of targets, then a colon, then a list of prerequisite files. Text following a semicolon, and all following lines that begin with a tab, are shell commands to be executed to update the target.

Pound sign (#) and newline surround comments.

The following makefile says that pgm depends on two files a.o and b.o, and that they in turn depend on It gives the rule for making pgm from a.o and b.o, and the rules for making a.o and b.o from a.s and b.s.

```
pgm: a.o b.o
```

```
        ld a.o b.o -lm -o pgm
a.o: incl a.s
        as -o a.o a.s
b.o: incl b.s
        as -o b.o b.s
```

Makefile entries of the form

```
string1 = string2
```

are macro definitions. Subsequent appearances of $(string1) are replaced by string2. If string1 is a single character, the parentheses are optional.

The **make** command infers prerequisites for files for which makefile gives no construction commands. For example, a and be assembled to produce the .o file. Thus the preceding example can be done more briefly:

```
pgm: a.o b.o
        ld a.o b.o -lm -o pgm
a.o b.o: incl
```

The **make** program itself does not know what file name suffixes are interesting or how to transform a file with one suffix into a file with another suffix. This information is stored in an internal table that has the form of a description file. If the **-r** parameter is used, this table is not used.

A user may add new suffixes and change or add default rules to his description files. The format of the suffix list and the transformation rules follow.

The list of suffixes is actually the dependency list for the name "SUFFIXES"; **make** looks for a file with any of the suffixes on the list. If such a file exists, and if there is a transformation rule for that combination, **make** acts as described earlier.

The transformation rule names are the concatenation of the two suffixes. The name of the rule to transform a .r file to a .o file is thus .r.o. If the rule is present and no explicit command sequence has been given in the user's description files, the command sequence for the rule .r.o is used. If a command is generated by using one of these suffixing rules, the macro **$*** is given the value of the stem (everything except the suffix) of the name of the file to be made, **$@** is the full name of the file to be made, **$<** is the list of the prerequisites, and **$?** is the list of prerequisites that are out of date.

The order of the suffix list is significant, since it is scanned from left to right, and the first name that is formed that has both a file and a rule associated with it is used. If new names are to be appended, the user can just add an entry for "SUFFIXES" in his own description file; the dependents will be added to the usual list. A "SUFFIXES" line without any dependents

deletes the current list. (It is necessary to clear the current list if the order of names is to be changed).

Command lines are executed one at a time, each by its own shell. A line is printed when it is executed unless the special target ".SILENT" is in makefile, or the first character of the command is "@".

Commands returning nonzero status cause **make** to terminate unless the special target ".IGNORE" is in makefile or the command begins with <tab><hyphen>.

Interrupt and quit cause the target to be deleted unless the target depends on the special name '.PRECIOUS'.

FILES
makefile
Makefile

SEE ALSO
See the discussion on **make** in the **8560 Series MUSDU System Users Manual.**

sh
touch

NOTES
Some commands return nonzero status inappropriately. Use the **-i** parameter to overcome the difficulty, or start that particular command with <tab><hyphen>.

Commands that are directly executed by the shell, notably **cd**, are ineffectual across newlines in **make.**

makekey -- generate encryption key

EXAMPLES

/usr/lib/makekey Invokes **makekey**.

SYNTAX
/usr/lib/makekey

EXPLANATION
The makekey command improves the usefulness of encryption schemes that depend on a key by increasing the amount of time required to search the key space. It reads 10 bytes from standard input and writes 13 bytes to standard output. The output depends on the input in a way intended to be difficult to compute (i.e. to require a substantial fraction of a second).

The first eight input bytes (the input key) can be arbitrary ASCII characters. The last two (the **salt**) are best chosen from the set of digits, uppercase and lowercase letters, and "." and "/". The salt characters are repeated as the first two output characters. The remaining 11 output characters are chosen from the same set the salt characters come from. These 11 characters constitute the output key.

The salt is used to select one of 4096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but modified in 4096 different ways. Using the input key as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 useful key bits in the result.

The makekey command is intended for programs that perform encryption, such as ed. Its input and output are usually pipes.

SEE ALSO
ed

man -- display information from online manual

EXAMPLES

man mv Causes an explanation of the **mv** command to be displayed on your terminal.

man make diff | lp1r
Prints explanations of the **make** and **diff** commands on the line printer.

SYNTAX
man [chapter] [title...]

PARAMETERS

chapter A section (1 -- 8) of the online manual. Standard TNIX commands are contained in Section 1 (the default). Section 2 contains information about system calls. Section 3 contains information about subroutines. Section 4 contains information about TNIX special files. Section 5 contains information about TNIX file formats. Section 6 contains information about 8540 commands and other optional commands. Sections 7 and 8 are usually empty.

title The name of the command or topic that is documented online. Long titles are truncated to 14 characters. Capitalization is ignored.

EXPLANATION
The **man** command prints an explanation (commonly called a "MAN page") of each of the specified commands or topics.

FILES
/usr/man/cat?/*

SEE ALSO
help
index

mesg -- permit or deny messages

EXAMPLES

mesg Reports whether or not messages can be received.

mesg -n Disables permission for messages to be received.

mesg -y Enables permission for messages to be received.

SYNTAX

```
     [-n]
mesg [-y]
```

PARAMETERS

-n no -- Disables write permission to your terminal by all other users.

-y yes -- Enables write permission to your terminal by all other users.

EXPLANATION

Without any parameters, **mesg** reports the current message permission state of your terminal. One of the following messages will be displayed:

is n Means there is no write permission to your terminal by all other users.

is y Means yes, there is write permission to your terminal by all other users.

With parameters, **mesg** disables or enables write permission to your terminal by all other users.

FILES

/dev/tty*
/dev

SEE ALSO

write

EXIT STATUS

0 Messages are receivable.

1 Messages are not receivable.

2 Error.

mkboot -- copy a boot block to the fixed disk in standalone mode

EXAMPLES

```
fbr filename to boot> mkboot
```

In this example, the **mkboot** command on the Standalone Utilities disk copies a new boot block onto the 8560's internal fixed disk.

SYNTAX

mkboot

EXPLANATION

The **mkboot** command installs the boot block in block 0 of the 8560's internal fixed disk. The boot block contains a short program that loads TNIX from the disk into system memory.

The **mkboot** command resides on the Standalone Utilities flexible disk, so you must place the system in standalone mode before reinstalling the boot block. When **mkboot** executes, it searches the flexible disk for a file named pmuboot. If this file exists, **mkboot** copies its first block to block 0 of the fixed disk.

SEE ALSO

For further information on **mkboot**, see the 8560 **Series System Managers Guide.**

fbr
format

ERROR MESSAGES

`cannot find pmuboot'

The **mkboot** command issues this message if the boot file is not on the Standalone Utilities flexible disk. Other error messages are similar to those generated by **fbr.**

mkdir -- make (create) a directory

EXAMPLES

mkdir zformat Creates directory zformat in your current directory.

mkdir /usr/jonb/testfiles

Creates the directory testfiles in the directory of /usr/jonb.

SYNTAX
mkdir dirname...

PARAMETERS

dirname The pathname of a **new** directory.

EXPLANATION
The **mkdir** command is used to create new directories in mode 0775 (octal--readable by all, writable by user and group, and searchable by all). These default permissions can be changed by changing the value of the file creation mode mask, which normally defaults to 2. See **umask** in Section 2 of this manual.

To check what the default mode is, type the command **umask** with no parameters. An octal number will be returned. Take the base 2 complement of the number and you have the directory mode. For instance:

```
umask   [typed by you]
2       [response from umask]
```

The complement of 002 (octal) is 775 (octal). In base 2, the complement is:

```
111 111 101
rwx rwx rwx
 u   g   o
```

This directly correlates to the mode. Each bit that is high (1) enables a permission (read, write, executable) for the user, group, or other. Each bit that is low (0) disables that permission. In this case, the only permission that is disabled is write permission for all users outside of your group (others).

The standard entries "." (for the directory itself) and ".." (for its parent directory) are made automatically for each specified directory.

The **mkdir** command requires write permission in the parent directory.

SEE ALSO
For information on the **umask** system call, see **umask** in Section 2 of this manual.

chmod
rm

EXIT STATUS
The **mkdir** command returns the following codes:

0	All directories were successfully made.
non-zero	Unsuccessful operation. A diagnostic message is printed and a non-zero number is returned.

mkfs -- construct a file system

EXAMPLES

CAUTION

The **mkfs** command destroys any data on the specified disk. Be sure you have backup copies of any valuable data before using **mkfs**.

```
fbr filename to boot> mkfs
mkfs arguments: 65000
```

In this example, the **mkfs** command on the Standalone Utilities disk creates a root filesystem on the 8560's internal fixed disk (/dev/hd0). The argument specified for **mkfs** indicates the number of disk blocks available for the root filesystem.

```
fbr filename to boot> mkfs
mkfs arguments: 24360
```

In this example, the **mkfs** command on the Standalone Utilities disk creates a root filesystem on the 8561's internal fixed disk (/dev/hd0). The argument specified for **mkfs** indicates the number of disk blocks available for the root filesystem.

```
fbr filename to boot> mkfs
mkfs arguments: 134576
```

In this example, the **mkfs** command on the Standalone Utilities disk creates a root filesystem on the 8560's internal fixed disk, and on an 8503 Disk Expansion Unit (/dev/hd01). The argument specified for **mkfs** indicates the number of disk blocks available for the root filesystem.

```
# mkfs /dev/hd1 69576
```

In this example, the standard TNIX **mkfs** command creates a filesystem on an 8503 Disk Expansion Unit (/dev/hd1). The system must be in multi-user mode when you use this command.

SYNTAX

mkfs (Standalone Utilities version)

or

mkfs logical-device blocks (standard TNIX version)

PARAMETERS

logical-device
: The special file name associated with the physical disk drive. For example, /dev/hd1 is the name of a logical device associated with a non-root filesystem on an 8503 Disk Expansion Unit.

blocks
: The number of blocks (decimal) on the logical device. (The Standalone Utilities version of **mkfs** also expects you to specify the number of blocks. Enter this parameter when standalone **mkfs** prompts you for arguments.) Table 1-7 lists the number of blocks for various filesystem and logical device configurations:

Table 1-7
Filesystem and Logical Device Block Capacities

Capacity of Each Disk Drive (in M Bytes)	Number of Disk Drives in Filesystem	Total Number of Blocks In Filesystem	
		If Root Filesystem	If Non-Root Filesystem
13.6	1	24360 (/dev/hd0)	not supported
35.6	1	65000 (/dev/hd0)	69576 (/dev/hd1, /dev/hd2, or /dev/hd3)
35.6	2	134576 (/dev/hd01)	139576 (/dev/hd12 or /dev/hd23)
35.6	3	204152 (/dev/hd02)	208728 (/dev/hd13)
35.6	4	273728 (/dev/hd03)	not supported

EXPLANATION

The **mkfs** command:

- Constructs an empty filesystem on the specified logical-device.
- Creates a default number of inodes in the filesystem, based upon the number of logical-device blocks available.

The Standalone Utilities version of **mkfs** is used to create the root filesystem (the filesystem that begins on fixed disk drive 0). You must boot the system from the Standalone Utilities flexible disk to use this version of **mkfs**.

NOTE

If you change the size of the root filesystem, you must reconfigure your system with the **cvt** program.

The standard TNIX version of **mkfs** is used to create any non-root filesystem. Be sure to unmount the filesystem first.

SEE ALSO
8560 Series System Managers Guide

For information on the **dir** file format, see **dir** in Section 5 of this manual.

For information on the **filsys** file format, see **filsys** in Section 5 of this manual.

cvt
mkboot
restor
syschk

mkgpib -- make, modify, or print GPIB device definitions

SYNTAX

mkgpib [**-q**] [*device*] [Check and print device definition.]

mkgpib -r [**-q**] *device* [Remove device.]

mkgpib -m [**-q**] [*device* [**-a** *primary*[.*secondary*]]
[**-d** *GDname*] [**-c** *comment*] [**-s** *option*=*value* [:...]]

[Install or modify device,
or correct device definition.]

mkgpib -i [**-q**] [Initialize GPIB Interface.]

PARAMETERS

device Specifies the complete pathname of a GPIB device. The pathname must always begin with */dev/*. For example, a TNIX magnetic tape drive name is often given a name such as */dev/mt1*.

-a *primary*[.*secondary*]
Specifies the device's GPIB address. *primary* is the device's primary address, and must always be included with the **-a** parameter. *secondary* represents the device's optional secondary address. Primary addresses may be any decimal integer from 0--30, and secondary addresses may be any decimal integer from 0--31.

-c *comment*
Indicates that the following *comment* is a brief description of the device. *comment* may not be longer than 80 characters, and cannot contain colons. If the comment contains characters that have special meaning to the shell (including spaces), these characters must be preceded by a backslash (\). As an alternative, the entire comment can be enclosed in single quotes (').

-d *GDname* Indicates that the device driver routine is *GDname* (GD stands for GPIB Instrument Driver). The GD converts TNIX I/O calls into the corresponding GPIB commands for a particular device. The following GD names are currently defined:

GD Name	Description
magtape	A device driver for controlling magnetic tape controllers manufactured by Dylon Corporation.
direct	A general-purpose device driver that allows low-level access to both GPIB-compatible instruments and the GPIB itself
instr	A general-purpose device driver that provides higher-level access to GPIB-compatible instruments than the "direct" GD.

See **gpib** in Section 4 of this manual for more information on these GDs.

-m Modify the /etc/gpib/dev device definition file.

If you omit device, **mkgpib** checks the integrity of all device definitions, and attempts to correct any errors. **mkgpib** then displays a list of device definitions and reports any uncorrectable errors.

If you specify device without further options, the check and correction process applies to that device only.

If you specify device with other options, **mkgpib** either adds or modifies device according to the subsequent options. If you're adding device, you must include both the **-a** and **-d** parameters.

-i Initialize the GPIB Interface, setting the internal device table to the contents of /etc/gpib/dev. If a particular definition in the file is corrupt, **mkgpib** displays an error message and ignores that device. Normally, **mkgpib -i** is only used by the **ldgpib** command during system initialization.

-q "Quiet" output. **mkgpib** does not display device definitions. However, **mkgpib** still reports any device definition errors.

-r Removes the specified device. **mkgpib** checks the device definition before removing the device, and informs you of any errors.

-s option=value[:...]
Sets a device-dependent option to value. You may specify values for several options by separating each value from the following option with a colon (:). The available options depend on the device's GD; not all GDs have options associated with them. Currently, the valid options and values are:

Options	Associated GD	Description	Values
type	magtape	Type of device.	9track = Dylon 2001/9001 7/9 track Magnetic Tape System dc-300 = Dylon 4000A Digital Cartridge Recording System
len	magtape	Tape record length (in bytes).	The value may be be any integer less than or equal to the maximum buffer size of the Dylon controller.

EXPLANATION

The **mkgpib** command is a general utility program for managing all GPIB-compatible devices on the system. Specifically, **mkgpib**:

- allows you to install or remove GPIB devices
- allows you to modify the definition of an existing GPIB device
- prints information about any (or all) device configuration
- checks the integrity of device definitions, and corrects errors when possible

The /etc/gpib/dev file contains the device definitions maintained by **mkgpib**.

EXAMPLES

Let's assume that you have just joined a new department in your company. The department has an 8560 MUSDU, and you're the new system manager.

Your department's 8560 has several options installed, including a GPIB Interface.

Adding a New Device

The first major task you've been assigned is to add a Dylon 9001 9-track tape drive to the system. The drive has 4K bytes of buffer memory. You decide that the drive will reside at GPIB address 3, and that the drive's device name will be "mt0". After you physically connect the drive to the GPIB, you must install the device on the system with this command:

```
# mkgpib -m /dev/mt0 -a 3 -d magtape -c '9-track tape'
-s type=9track:len=4096
mt0             magtape   3.    type=9track:len=4096 9-track tape
```

Modifying Device Definitions

About a week later, your boss decides that the tape drive should be located at GPIB address 1. All you have to do to make the change is reset the drive's GPIB switches, and enter this command:

```
# mkgpib -m /dev/mt0 -a 1
mt0             magtape   1.    type=9track:len=4096 9-track tape
```

Listing and Checking Device Definitions

At this point, it may be useful to review the system GPIB configuration. You can use **mkgpib** without arguments to display a list of all installed GPIB devices, and their configurations:

```
# mkgpib
mt0             magtape   1.    type=9track:len=4096 9-track tape
```

If you want to find out about an individual device (for example, mt0), you can follow **mkgpib** with the device name:

```
# mkgpib /dev/mt0
mt0             magtape   1.    type=9track:len=4096 9-track tape
```

The preceding two forms of **mkgpib** also check the integrity of the device definitions. Two sets of device definitions exist: one maintained by the 8560's system processor, and the other by the microprocessor-based GPIB Interface. **mkgpib**'s device definition check ensures that these two definition sets match. In the event that one set becomes corrupted, **mkgpib** detects the inconsistency and report an error, such as:

incorrect GD defined on GPIB for /dev/counter

You can check all device definitions without receiving a listing by including the **-q** parameter with **mkgpib**:

```
# mkgpib -q
```

If **mkgpib** detects any inconsistencies, it will display an error message. Otherwise, **mkgpib** doesn't create any output; your shell prompt returns when the command finishes.

Checking and Correcting Device Definitions

If you use **mkgpib -m** without further parameters, **mkgpib** checks the integrity of all device definitions **and** attempts to correct any inconsistencies. For example, if errors were reported when you executed **mkgpib** without arguments, you can enter:

```
# mkgpib -m
```

If **mkgpib** can't correct the problem, it will say so.

If you want to check and correct the definition of just one device, you can enter the preceding command, followed by the device name. For example:

```
# mkgpib -m /dev/mt0
```

Normally, inconsistencies will not occur between the two sets of device definitions. If errors of this sort occur often, a software or hardware malfunction is probably responsible; contact your Tektronix field service representative for help.

FILES

/dev/gpib	The 8560 GPIB Interface device.
/dev/???	Various logical GPIB-compatible devices. These file names must not conflict with existing TNIX special file names, such as /dev/kmem or /dev/hd0.
/etc/gpib/dev	Contains the definitions of all GPIB-compatible devices on the system.
/etc/gpib/gd	Contains each GD name and an associated TNIX identification number.
/tmp/mkgpib.lock	Exists when a **mkgpib -m** or **mkgpib -r** command is executing.

SEE ALSO
8560 **Magnetic Tape Interface Users Manual.**

For information on the **gpib** file, see **gpib** in Section 4 of this manual.
For information on the **gpib** file format, see **gpib** in Section 5 of this manual.

ldgpib
settape
statgpib

NOTES
The **mkgpib** command does not verify that you've specified valid options with the **-s** parameter. However, other utilities (such as **settape**) check the validity of relevant options and report errors if the options are invalid.

Comments are truncated to 25 characters in displays, and to 80 characters in the /etc/gpib/dev file. Comments cannot contain colons.

You cannot modify a device with **mkgpib** while the device is in use.

mkgpib maintains the interface between TNIX and the GPIB Interface hardware, but it does not affect instruments connected to the GPIB.

If an inconsistency is detected in a device definition, **mkgpib** always assumes that the definition in /etc/gpib/dev is correct and conforms the definition on the GPIB Interface hardware to this definition.

mkgroup -- add new group to system

EXAMPLES

mkgroup assembler jackr lorii root

Creates a new group of the name assembler with three users (jackr, lorii, and root) as members of the group.

SYNTAX
mkgroup groupname username...
mkgroup -r groupname

PARAMETERS

groupname The name of the new group.

username One of the members of the group.

-r Causes the the specified group to be removed from the system.

EXPLANATION
This command adds the new groupname to the system and makes each username a member of that group. The **-r** parameter removes the named groupname from the system.

ERROR MESSAGES

group name already exists The group name that you have specified is already in use.

username **does not exist** The user name that you have specified is already in use.

SEE ALSO
For information on the **group** file format, see **group** in Section 5 of this manual.

For information on the **passwd** file format, see **passwd** in Section 5 of this manual.

mkuser

NOTES
To use the **mkgroup** command, you must be logged in as root.

The **mkgroup** command needs to open the passwd and group files exclusively.

mkuser -- create or modify a user's account

EXAMPLES

mkuser lazarusl z80 hyperion

Creates an account for "lazarusl", assigning lazarusl to groups "z80" and "hyperion".

mkuser -r lazarusl hyperion

Removes "lazarusl" from group "hyperion".

mkuser -r lazarusl

Deletes lazarusl's user account from the /etc/passwd file.

SYNTAX
mkuser [**-r**] username [groupname]...

PARAMETERS

-r Causes a removal of the specified user or group name from the system.

username The user name of an account.

groupname The group name that username is to be assigned to.

EXPLANATION
The **mkuser** command creates or modifies an account for username.

If an account for username does not exist, **mkuser** creates an account for username on the system and assigns this user to the groupnames that you specify. If you don't specify a groupname to assign username to, **user** is assigned to the "default" group. The default login directory /usr/username is created, and no password is initially assigned to this account. Also, the file /usr/lib/default.profile is copied to the user's login directory, /usr/username.

If an account for username already exists, the default group and comment field entries for that user (found in the /etc/passwd file) are displayed. You can change these parameters by entering a new value for them, or you can press the RETURN key to leave these values unchanged.

The **-r** parameter removes username from the named groupnames without modifying the /etc/password file. If you don't specify a groupname to remove username from, username's account is deleted.

FILES
/etc/passwd
/etc/group
/etc/grplock
/tmp/pwdXXXXXX
/usr/lib/default.profile

SEE ALSO
For information on the **group** file format, see **group** in Section 5 of this manual.

mkgroup
passwd
setuser

NOTES
To use the **mkuser** command, you must be logged in as root.

The **mkuser** command must open the /etc/passwd and /etc/group files exclusively.

ERROR MESSAGES

group 'groupname' does not exist

The group name that you specified (groupname) does not exist.

mload -- upload/download Tekhex files

EXAMPLES

DOWNLOAD A PROGRAM FROM TNIX TO THE 8540 (IN HEX FORMAT)

To transfer a program from an 8560 to a Tektronix 8540, enter the 8540 COM command, log into TNIX, and enter the following command:

```
mload -d tnix-to-8540 <CR>
```

Then enter a null character and press the return key:

```
<CTRL-@> <CR>
```

This procedure causes the contents of the TNIX file, tnix-to-8540, to be transferred into the 8540's memory.

UPLOAD A PROGRAM FROM THE 8540 TO TNIX (IN HEX FORMAT)

To transfer a program from a Tektronix 8540 to an 8560, enter the 8540 COM command, log into TNIX, and enter the following command:

```
mload -u 8540-to-tnix <CR>
```

To tell the 8540 where the program is located, enter a null character immediately followed by the beginning and ending address of the program:

```
<CTRL-@>0 0FFF <CR>
```

This causes a copy of memory locations 0000-0FFF (hexadecimal) to be transferred into the TNIX file 8540-to-tnix.

SYNTAX

mload [**-d**] [**-s**] [**-p** *prompt-string*] [**-u**] *name*

PARAMETERS

-u upload -- Directs **mload** to upload a program from the 8540/8550/8001/8002 to the 8560 and store it in the TNIX file *name*.

-d download -- Directs **mload** to download a program from the TNIX file *name* into 8540/8550/8001/8002 memory. (This is the default parameter.)

-s Automatically selects the transfer rate required by a Tektronix 8001 or 8002.

-p prompt-string
A (literal) prompt string. This parameter is ordinarily omitted. Include it **only** of you include a corresponding P parameter in the COM command line.

name The pathname of a TNIX file containing (or to contain) the formatted memory image (the program).

<CTRL-@>
The ASCII **NUL** character. This is typed in by holding the control (CTRL) key down while pressing the **@** key once. Do not type in the "<" and ">" characters.

EXPLANATION
The **mload** command provides the host side of the COM protocol with formatted data transfers between TNIX and an 8001, 8002, 8540, or 8550. The following hex formats are currently supported:

- Standard TEKHEX
- Extended TEKHEX
- Intel hex
- Motorola hex

SEE ALSO
uload

The "Intersystem Communication" section in the **8540 System Users Manual.**

NOTES
Prompt sequences (**-p** parameter) that include shell characters (such as the dollar sign, **$**) must be enclosed within apostrophes (**'**). For instance, to specify the prompt **$$**, you would enter **-p '$$'** within the **mload** command line.

The echo mode (set with the **stty** command) is disabled during the transfer.

The interrupt signal, **<CTRL-C>**, aborts the transfer.

more -- display a screenful at a time

EXAMPLES
more -s filea fileb

This command will display the two files filea and fileb on the terminal, one screenful at a time. The -s parameter causes any group of multiple blank lines in the files to be displayed as a single blank line.

SYNTAX
more [-c][-d][-f][-l][-s][-u][-n] [+linenumber] [+/pattern] [filename...]

PARAMETERS
The command line parameters are:

-n Sets display window to n (an integer) lines. The default is taken from the termcap definition of the terminal.

-c Turns off scrolling. This parameter will be ignored if the terminal cannot clear to the end of a line.

-d Prompts with the message: "Hit space to continue, Rubout to abort" at the end of each screenful.

-f Displays logical rather than screen lines. That is, long lines (lines with more characters than the screen width) are not folded. This parameter is used if the text in the file has escape sequences that do not print on the terminal.

-l Ignores ^L (form feed). The default setting (no -l) treats a ^L as if the end of a screenful had been reached. If a file begins with a form feed, **more** always will clear the screen before the file is printed.

-s Squeezes multiple blank lines from the output, displaying only one blank line for each series of multiple blank lines. Using this parameter, multiple empty lines in text files are not displayed.

-u Suppresses underlining and standout mode when displaying text if the terminal has these options.

+linenumber
Starts display at linenumber.

+/pattern
Starts display two lines before the line containing the regular expression pattern. Enter **man ed** for an explanation of regular expressions.

filename
The pathname of a file.

EXPLANATION

The **more** command acts as a filter to allow you to examine text on a terminal (from a file or a pipe) one screenful at a time. After each screenful, **more** pauses, printing the following prompt at the bottom of the screen:

```
--More-- (N%)
```

Entering a carriage return causes the next line to be displayed. Entering a space causes another whole screenful to be displayed.

The "(N%)" in the prompt tells you what percentage of the file (in characters) has been read. If **more** is reading from a pipe the percentage is not displayed. N is an integer from 0--100.

If the output from **more** is redirected to a device other than a terminal, **more** simply transmits the file, including a header with the file name if more than one file is being read.

THE MORE ENVIRONMENT

The environment variable MORE may be set with options for **more**. For example the command **MORE=-cp; export MORE** causes **more** to use the **-c** and **-p** parameters when it is invoked.

The **more** command looks in file /etc/termcap to determine terminal characteristics and to determine the default window size.

more Commands

The **more** command accepts commands, which may be entered when **more** pauses after displaying each screenful of text. These commands do not require a carriage return.

In the following commands, N is an optional integer argument that defaults to 1.

The commands are:

<delete> (rubout)
Exit **more**.

<return>
Display the next line.

<space>
Display another screenful.

N<space>
Display N more lines.

^\(quit character)
Entering this command during terminal display (before the **more** prompt is normally displayed) stops the display, issues the prompt, and permits you to enter commands. This command results in the loss of some lines of the display.

^D Scroll 11 more lines.

^H (erase character)
Redisplay the **more** prompt.

^U (kill line character)
Cancel command. This cannot be used on commands that are one character long.

d Same as ^D (control-D).

[N]z Display the next N lines and set the window size to N. If N is not present, display the next screenful.

Ns Skip N lines and print a screenful of lines.

Nf Skip N screenfuls and print a screenful of lines.

q or Q
Exit from **more**.

= Display the current line number.

N/expr
Search for the Nth occurrence of the regular expression expr. If there are less than N occurrences of expr, and the input is a file rather than a pipe, then the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found. The backspace and kill line characters may be used to edit the regular expression. Backspacing past the first column cancels the search command.

Nn Search for the Nth occurrence of the last regular expression entered.

' (single quote)
Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!command
Invoke a shell with command. The characters '%' and '!' in "command" are replaced with the current file name and the previous shell command, respectively. If there is no current file name, '%' is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!", respectively.

N:n Skip to the Nth file given in the command line. Skips to last file if n doesn't make sense.

N:p Skip to the Nth previous file given in the command line. If this command is given during the display of a file, then **more** goes back to the beginning of the file. If N doesn't make sense, **more** skips back to the first file.

:f Display the current file name and line number.

:q or :Q
Exit from **more**.

. (period)
Repeat the previous command.

FILES

/etc/termcap Terminal data base.

mount, umount -- mount and unmount filesystems

EXAMPLES

/etc/mount /dev/hd0 /usr1

Mounts logical-device /dev/hd0 for use as the filesystem called /usr1. All files on logical-device "/dev/hd0" are located in a subdirectory of "/usr1".

/etc/umount /usr1

Unmount the /usr1 filesystem.

SYNTAX

/etc/mount [logical-device filesystem [**-r**]]

/etc/umount logical-device

PARAMETERS

logical-device The name of the logical device that you are going to mount.

filesystem The name of the newly mounted root. This should also be a directory name.

-r Specifies the mounted filesystem to be in read-only mode.

EXPLANATION

The **mount** command is used to create a new filesystem on the 8503 Disk Expansion Unit. The **mount** command tells TNIX that a filesystem, located on the specified logical-device, is present on the 8560. Data can be written to and read from filesystem after it is mounted. The filesystem must exist **before** using the **mount** command.

The filesystem must be a directory (unless the root of the mounted filesystem is not a directory) and becomes the name of the newly mounted root.

If **mount** is invoked without any parameters, it prints the table of currently mounted filesystems (except for the root filesystem) available to the **mount** and **umount** commands.

The **umount** command announces to the system that the removable filesystem previously mounted on logical-device filesystem is to be removed.

FILES

/etc/mtab table of mountable logical-devices

SEE ALSO

For information on the **mount** system call, see **mount** in Section 5 of this manual.

For information on the **mount** system call, see **mount** in Section 5 of this manual.

mtab mtab

NOTES

Flexible disks can be mounted as filesystems. Use the **mkfs** command to format the disk.

Physically write-protected filesystems must be mounted read-only (using the **-r** parameter). If this is not done, errors occur when access times are updated, whether or not an explicit write is attempted.

Mounting a logical-device that does not contain a valid filesystem will **crash** the system.

mv -- move or rename file or directory

EXAMPLES

mv develop/parser.c archive	Moves file parser.c from directory develop to directory archive.

SYNTAX

mv file1 file2

mv file... directory

mv directory1 directory2

PARAMETERS

file	The pathname of a file.
directory	The pathname of a directory.

EXPLANATION

The **mv** command moves (changes the pathname of) file1 to file2.

If file2 already exists, it is deleted before file1 is moved. If file2's mode forbids the deletion, **mv** prints the mode (see **chmod**) and asks if file2 may be deleted. If you enter **y**, the move takes place; if not, **mv** terminates.

In the second syntax form, one or more files are moved to the specified directory, retaining their original filenames.

In the third syntax form, two different things can happen.

> If directory2 already exists, directory1 becomes a subdirectory of directory2.
>
> If directory2 does not exist, then the effect is simply to change the pathname of directory1 to directory2.

SEE ALSO

chmod
cp

NOTES

The **mv** command refuses to move a file onto itself.

If file1 and file2 lie on different filesystems, **mv** copies the file and deletes the original. The owner name becomes that of the copying process and links to other files are lost.

newgrp -- log in to a new group

EXAMPLES

newgrp wrap Changes your current group to "wrap", as long as you are a member of the wrap group.

SYNTAX
newgrp group

PARAMETERS

group The name of the group under which you want to reclassify yourself.

EXPLANATION
The **newgrp** command changes your current group ID. You remain logged in, and your current directory is unchanged, but file access permissions are determined with respect to the new group ID.

When you log in, you are a member of the group specified by the fourth item in the password file, /etc/passwd.

FILES
/etc/group
/etc/passwd

SEE ALSO
For information on the **group** file format, see **group** in Section 5 of this manual.

chmod
login
mkgroup

nice, nohup -- run a command at low priority

EXAMPLES

nice -20 runprogram	Causes the process runprogram to be run at the lowest possible priority, specified by -20.

SYNTAX

nice [-number] command

nohup command

PARAMETERS

-number The priority number.

command The command to be executed by **nice**.

EXPLANATION

The **nice** command executes command with low scheduling priority. If the number argument is present, the priority is incremented by that amount up to 20. Higher numbers signify lower priorities. The default for number is 10.

The superuser may run commands with higher than normal priority by using negative priorities (such as "--10").

The **nohup** command executes command immune to hangup and terminate signals from the controlling terminal. The priority is incremented by 5. The **nohup** command should be invoked from the shell with "&" to prevent it from responding to interrupts, or stealing the input from the next person who logs in on the same terminal.

FILES

nohup.out The standard output and standard error file under nohup.

SEE ALSO

For information on the **nice** system call, see **nice** in Section 2 of this manual.

EXIT STATUS

The nice command returns the exit status of the subject command.

od -- display file in octal format (octal dump)

SYNTAX

od [**-bcdox**] [**-s** *offset*[**.**] [**b**]] [**-e** *offset*[**.**] [*file*]

PARAMETERS

-b Interpret bytes in octal.

-c Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes:

- null=\0
- backspace=\b
- formfeed=\f
- newline=\n
- return=\r
- tab=\t

Other non-graphic characters appear as 3-digit octal numbers.

-d Interpret words in decimal.

-o Interpret words in octal. This is the default interpretation.

-x Interpret words in hexadecimal.

-s *offset*
The *offset* argument specifies the offset in the file where dumping is to begin. This argument is normally interpreted in octal bytes. If "." is appended, the offset is interpreted in decimal. If "b" is appended, the offset is interpreted in blocks of 512 bytes.

-e *offset*
Specifies where dumping is to end. The default is dumping until end-of-file.

file The pathname of the file to be dumped.

EXPLANATION

The **od** command dumps file in one or more formats, as selected by the flags. If no flags are present, **-o** is the default.

The file argument specifies which file is to be dumped. If no file argument is specified, standard input is used.

A dump line consisting of a lone "*" indicates repetition of the previously dumped data.

passwd -- change login password

EXAMPLES

```
passwd          [the  password is entered  after the  command, but
                [is not visible (not echoed back) on your terminal]
Old password:   [enter  your old  password here; it will not be
                 echoed back to your terminal]
Changing password for janer
New password:   [enter your new password here; it will not be
                 echoed back to your terminal either]

Retype new password:     [here, you must enter the new password
                          again, just in case you  misstyped it]
```

Changes the current password to whatever you specify it to be.

SYNTAX

passwd [name]

PARAMETERS

name The character sequence of the new password you want to use for your account.

EXPLANATION

This command changes or installs a password associated with the user name (your own name by default).

The program prompts for the old password and then for the new one. The caller must supply both passwords. The new password must be typed twice, to forestall mistakes.

Only the owner of the name or the superuser may change a password; the owner must prove he knows the old password by supplying it.

FILES

/etc/passwd

SEE ALSO

For information on the **crypt** subroutine, see **crypt** in Section 3 of this manual.

For information on the **passwd** file format, see **passwd** in Section 5 of this manual.

login

ERROR MESSAGES

Sorry Displayed if you enter the wrong old password.

Mismatch - password unchanged.
Displayed if you do not enter the new password exactly the same both times.

pr -- format a file

EXAMPLES

pr -p 3 printme

Prints from page 3 through to the end of file printme.

man dsc50 | pr | lp1r

Prints the online manual page for the **dsc50** command on the lineprinter in a paged format.

SYNTAX

pr [**-ntm**] [**-c** *k*] [**-l** *k*] [**-p** *k*] [**-s** *c*] [**-w** *k*]
[**-h** *header*] [*file*...]

PARAMETERS

-n Adds line numbers.

-t Disables the 5-line header and the 5-line trailer normally supplied for each page.

-m Prints all *files* simultaneously, each in one column.

-c *k* Produces *k*-column output.

-l *k* Specifies the line length of a page to be *k* lines. The default line length is 66 lines.

-p *k* Begins printing with page *k*.

-s *c* Separates columns by the single character *c*, instead of by the appropriate amount of white space.

-w *k* For purposes of multi-column output, specifies the page width to be *k* characters. The default page width is 72 characters.

-h *header*
Use *header* as the page header.

file The pathname of a file.

EXPLANATION

The **pr** command produces a printed listing of one or more *files*. The output is separated into pages headed by a date, the name of the file or a specified header, and the page number. If there are no file arguments, **pr** prints its standard input.

Inter-terminal messages through **write** are forbidden during a **pr**.

All parameters remain unchanged until modified again. Thus using **pr** without parameters causes **pr** to use the same parameters it used the last time it was invoked.

FILES

/dev/tty?	Used to suspend messages.

SEE ALSO
cat

ps -- show status of commands currently executing (process status)

EXAMPLES

ps	Displays all your currently executing commands (processes).
ps -xa	Displays all processes running on the system.

SYNTAX
ps [-alx]

PARAMETERS

-a Displays information about all processes associated with terminals, instead of only your own processes.

-l (Long listing.) Supplies additional information about currently executing processes.

-x Displays information about all processes, both those associated with terminals and those without terminals.

EXPLANATION
The **ps** command shows a "snapshot" of currently active processes. A process is any independently executing command or program with its own current directory, open files, user identification number, and other program-specific data. A single command may generate more than one process, such as a shell procedure, which in turn invokes shell commands.

The short display contains the fields:

```
PID TTY      TIME CMD
```

The long display contains the fields:

```
F S  UID   PID  PPID CPU PRI NICE  ADDR  SZ WCHAN  TTY      TIME CMD
```

The **ps** display fields contain:

F (Memory flags) Shows the memory status of the process.
- 01 = In memory.
- 02 = System process.
- 04 = Locked in memory (as in physical I/O).
- 10 = Being swapped between memory and disk.
- 20 = Being traced by another process.

S (State) Shows the state of the process.
I = Intermediate. Process is being created.
R = Running.
S = Sleeping (waiting for high priority interrupt, as in I/O).
W = Waiting (waiting for a low priority interrupt).
T = Stopped. Process is being traced, as in debugging.
Z = Terminated. Finished but not yet removed from memory.

UID The process owner's user identification number. This is found in the password file (/etc/passwd). You can use this number to find the login name of the person who started a process. For example, if the user ID associated with the process is 182, the command:

```
$ grep 182 /etc/passwd
```

returns user 182's entry in the password file. That user's login name is the first item in that entry (before the first colon).

PID Process identification number. You use this number if you want to kill one of the active processes (see **kill**). For example, to kill process 876 from the following process status:

```
$ ps
PID TTY      TIME COMMAND
802 kb       0:00 lde ps.w
272 kb       0:00 sh -i
876 kb       0:03 loan
```

you would enter the following **kill** command:

```
$ kill -9 876
```

PPID Process ID of the parent process.

CPU Number used in CPU scheduling.

PRI The priority of the process. Low numbers are high priority.

NICE Number used in assigning priority.

ADDR If the process is in core memory, ADDR represents the starting memory address; otherwise, it signifies the disk address.

SZ The size in blocks of the memory image (of the process).

WCHAN The event for which the process is waiting or sleeping. If blank, the process is running.

TTY Terminal (tty) number associated with the process.

TIME Sum of execution time so far (in CPU seconds).

CMD A breakdown of the command(s) that invoked the process.

Here's an example of the short (default) display:

```
$ man sh | lp1r
$ ps
PID TTY      TIME CMD
876 console  0:01 lp1r
```

In this example, a command to print the **sh** online manual page on the lineprinter generated the process. The process was given the arbitrary ID number 876 and was entered from the console terminal. The cumulative CPU time is 0:01 (one second).

Here's an example of the long display:

```
$ man sh | lp1r
$ ps -l
F S  UID    PID  PPID CPU PRI NICE  ADDR  SZ WCHAN  TTY       TIME CMD
1 S    9    883    56  32  26  20  3536  15 124646 console   0:01 lp1r
```

FILES

/unix	System namelist.
/dev/mem	Memory.
/usr/sys/core	Alternate memory file.
/dev	Searched for terminal tty numbers and swap device.

pwd -- display name of current (working) directory

SYNTAX
pwd

EXPLANATION
The **pwd** command prints the pathname of the directory you are working in. This is commonly known as the "current" or "working" directory. For example:

```
$ pwd
/usr/joet/tools
```

This tells you that your current directory is /usr/joet/tools.

SEE ALSO
cd

restor -- restore files from backups

EXAMPLES

restor -f /dev/rfd0 -t

Print the date and dump level of the dump volume in the flexible disk drive.

umount /usr1; restor -f /dev/rfd0 -r /usr1

Unmounts the /usr1 filesystem, then copies the contents of the dump volume(s) into the /usr1 filesystem.

fbr filename to boot> restor

restor arguments: -f mt(0,1)

Restores the root filesystem from a Dylon Corporation 2001/9001 7/9 track magnetic tape unit located at GPIB primary address 1. Execute this command while the system is in stand-alone mode.

SYNTAX

restor [**-f** logical-device] **-rxt** [filename...]

or

restor [**-f mt**({0}|{1}),addr)]

PARAMETERS

-f logical-device

Specifies that logical-device is the device containing the dump volume. The device name must begin with /dev/. If you don't specify logical-device, **restor** assumes that /dev/rfd0 contains the dump volume.

-r Writes the contents of the dump volume(s) into the filesystem specified by filename. This operation recreates a filesystem from either full or incremental backups. You should unmount filename before performing the restore operation. See **mount** for information about mounting and unmounting filesystems.

You cannot use the **-r** parameter to restore the root filesystem; you must use the standalone version of **restor** (described later) to restore the root filesystem from backups.

-x Extracts each filename from the dump volume, and places filename's contents into a file in the current directory. The resulting file is named with filename's inode number.

-t Print the date the volume was written and the date the filesystem was dumped.

EXPLANATION

The **restor** command is used to read flexible disks or magnetic tapes dumped with the **dump** command. (These flexible disks or magnetic tapes are known generically as dump volumes.) The parameter that you select specifies what action is to be taken.

RESTORING A COMPLETE FILESYSTEM

To restore a complete filesystem from backups, perform the following steps:

1. Restore the most recent complete (dump level 0) backup.

2. Restore the lowest-level incremental backup that is more recent than the level 0 backup. For example, if level 1, 2, and 3 backups have been performed since the latest level 0 backup, restore the most recent level 1 backup.

3. Restore the next-higher-level backup that is more recent than the previous incremental backup restored. Repeat this step until all incremental backups that meet this condition have been restored.

The following algorithm shows how to perform a restore operation:

```
level := 0;
restore(level);
last_level := level;

for level := 1 to 9 do
   begin
      if (date_of(level) > date_of(last_level)) then
         begin
            restore(level);
            last_level := level;
         end
   end
```

If you are restoring a non-root filesystem, the restore operation can be performed while TNIX is running. Otherwise, you must use the standalone **restor** program to restore the root filesystem.

RESTORING THE ROOT FILESYSTEM

There is a standalone version of **restor** available for restoring a complete or incremental dump to the root filesystem (the one that includes fixed disk drive 0). The standalone version is invoked by booting the "standalone utilities" disk and replying **restor** when prompted for the filename to boot.

When **restor** prompts you for arguments, it expects a carriage return if you are using flexible disks, or a response in the form **-f mt**(type,address) if

you are using magnetic tape. Type must be either a "0" (indicating a Dylon 2001/9001 7/9 track Magnetic Tape System) or a "1" (indicating a Dylon 4000A Digital Cartridge Recording System). address is the tape drive's primary address. For example, if you have Dylon 9-track tape drive at GPIB primary address 3, respond to the prompt with:

mt(0,3)

An appropriate warning message is issued by **restor**, before starting the restore operation, to allow the first volume to be installed. If more than one volume is involved in the dump, **restor** will tell you when to mount the next volume.

EXAMPLES

This example is a procedure for restoring files from either a full or an incremental backup.

Assume that you need to restore all files belonging to user bjorng. The files may be on any or all of three dump volumes (in this case, flexible disks). The dump volumes are labeled by inode numbers: volume I contains 2--1006, volume II contains 1007--4628, and volume III contains 4629--7268.

Your first task is to find all files belonging to bjorng on the dump set and to sort the filenames in order of increasing inode numbers. To make things easy, you can put the sorted list into the file list. One command line performs the task (be sure to put the **FIRST** dump volume into the drive):

```
$ dumpdir -f /dev/rfd0 | grep /usr/bjorng | sort +2 > list<CR>
```

The file list now contains the following text:

```
dumpdir:    /usr/bjorng/file.cabinet    237
dumpdir:    /usr/bjorng/file.single     516
dumpdir:    /usr/bjorng/file.rasp       2813
dumpdir:    /usr/bjorng/file.nail       2815
dumpdir:    /usr/bjorng/file.round      6972
```

According to list, the dump set contains five files that belong to bjorng. For each entry in list, the file's inode number follows the file's name.

Your next task is to divide the filenames in list among three new files, vol.1, vol.2, and vol.3. These new files correspond to the three dump volumes, and contain the names of bjorng's files that reside on each volume. That is, you want vol.1 to contain

```
/usr/bjorng/file.cabinet
/usr/bjorng/file.single
```

and vol.2 to contain:

```
/usr/bjorng/file.rasp
/usr/bjorng/file.nail
```

and vol.3 to contain:

/usr/bjorng/file.round

Use the following command lines to edit list and create vol.1, vol.2, and vol.3:

```
$ ed list<CR>
g/n.*\/usr/s//\/usr/<CR>
g/      .*$/s///g<CR>
w<CR>
1,2w!vol.1<CR>
3,4w!vol.2<CR>
5w!vol.3<CR>
q<CR>
$
```

Now, you can use vol.1, vol.2, and vol.3 with **restor** to write bjorng's files into the current directory. Install the first dump volume into the flexible disk drive and type:

```
$ restor -xf /dev/rfd0 `cat vol.1`<CR>
```

Install the second dump volume and type:

```
$ restor -xf /dev/rfd0 `cat vol.2`<CR>
```

Install the third dump volume and type:

```
$ restor -xf /dev/rfd0 `cat vol.3`<CR>
```

The current directory now contains bjorng's five files. You can move these files to whatever other directory you wish.

FILES

rst*	Temporary files used by **restor**.

SEE ALSO

8560 Series System Managers Guide

dump
dumpdir
mkfs
mount

NOTES

Restoring a complete dump to a filesystem (using a standalone **restor** for the root filesystem, or a "**restor -r ...**" to any other filesystem) destroys any information previously located on that filesystem.

ERROR MESSAGES

There are various diagnostics involved with reading the dump volume and writing the hard disk. There are also diagnostics if the i-list or the free list of the file system is not large enough to hold the dump.

If the process extends over more than one volume, **restor** will ask you to change volumes. Press RETURN when the next volume has been mounted.

rm, rmdir -- remove (unlink) files or directories

EXAMPLES

rm /usr/tmp/lur.o trot.c

Removes files /usr/tmp/lur.o and trot.c.

rm *.c

Removes all files in your current directory that end with the two letters ".c".

rmdir /usr/tmp/lloyd

Removes directory /usr/tmp/lloyd, but only if there are no files (except . and ..) or subdirectories in it.

SYNTAX

rm [-fri] file...

rmdir directory...

PARAMETERS

-f	force -- Deletes the file entry without question, whether or not there is no write permission.
-r	recursive -- Will recursively delete the entire contents of the specified directory, including the directory itself.
-i	interactive -- Causes **rm** to inquire whether or not to delete each specified file.
file	The pathname of the specified file.
directory	The pathname of the specified directory.

EXPLANATION

The **rm** command removes the **entries** for one or more files from the specified directory. Note that if an entry for a file was linked to another entry, you may still access the file, even though you removed one of its entries. But, if the entry removed is the **last** link to a file, the file itself is removed and is no longer accessible. The disk space that was taken up by the file is now freed for some other use.

The **rmdir** command removes entries for the named directories. A directory **must** be empty to be removed. You **must** use the **rmdir** command to remove a directory.

Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and standard input is from a terminal, its permissions are printed and a line is read from the terminal. If that line begins with "y", the entry for the file is deleted; otherwise it remains.

SEE ALSO
For information on the **unlink** system call, see **unlink** in Section 2 of this manual.

NOTES
You are not permitted to remove the "..." file. This restriction exists to save you from inadvertently entering a command like "**rm -r .***". This would remove all entries in your current directory that started with a period (.), including the "." and "..." files.

settape -- Set GPIB tape drive characteristics

EXAMPLES

settape /dev/mt2	Displays the device characteristics: device name, type of device, maximum record length, and rewind parameter.
settape /dev/mt3 -rq	Sets the rewind parameter to **yes** (which means to rewind the tape when the device is closed) and does not display the device characteristics.

SYNTAX

settape name... [-q] [-n] [-r]

PARAMETERS

name Specifies the complete pathname of the magnetic tape device. The pathname must always begin with /dev/. If you include more than one name, the **-n**, **-q**, and **-r** parameters apply to each device. The **settape** command checks each name before modifying any characteristics, to be sure that name specifies a GPIB-compatible device.

-q quiet; **settape** does not print out device characteristics.

-n Specifies that name does not rewind the tape when closed.

-r Specifies that name rewinds the tape when closed.

EXPLANATION

The **settape** command prepares the specified magnetic tape device for use, and displays the device's characteristics. These characteristics are of two types:

- Operational characteristics that determine whether or not the physical tape drive rewinds the tape when the device is closed.
- System characteristics: the device's name, the physical drive type, and the device's record length.

The **settape** command always resets all the system characteristics to their default values, which are contained in the /etc/gpib/dev file. The operational characteristics are modified by **settape** only when you specify the **-n** or **-r** parameters.

You should always execute **settape** before performing any magnetic tape operations to ensure that subsequent tape commands operate properly.

EXAMPLES

Assume that you want to use magnetic tape device /dev/mt1. Before you use this device, you use the **settape** command to reset the device's characteristics:

```
$ settape /dev/mt1 <CR>
```

The command returns the following display to your terminal:

```
DEVICE          TYPE      MAX RECORD  REWIND
/dev/mt1        9track    4096        no
```

Now let's assume that you want the tape drive to rewind the tape when the device is closed (that is, when subsequent tape commands finish and close the special file associated with the tape drive). You can enter the following command to set the rewind characteristic appropriately:

```
$ settape /dev/mt1 -r<CR>
```

The command returns the following display, verifying the change you just made:

```
DEVICE          TYPE      MAX RECORD  REWIND
/dev/mt1        9track    4096        yes
```

Once you finish using the tape drive, you may want to change the rewind characteristic back again (so the tape doesn't rewind when the device is closed). This time, you don't want to bother with the display, so you include the **-q** parameter. Your command line looks like this:

```
$ settape /dev/mt1 -q -n <CR>
```

FILES

/dev/gpib	The 8560 GPIB Interface device.
/etc/gpib/dev	Contains the definitions of all GPIB-compatible devices on the system.

SEE ALSO

8560 Magnetic Tape Interface Users Manual

For information on the **ioctl** system call, see **ioctl** in Section 2 of this manual.

For information on the **gpib** file, see **gpib** in Section 4 of this manual.

For information on the **gpib** file format, see **gpib** in Section 5 of this manual.

/etc/ldgpib
mkgpib

statgpib

NOTES

The **settape** command prints an error message if the tape drive is not on-line.

Also, the **settape** command does not have a default value associated with the **-n** and **-r** parameters. You must explicitly specify one of these parameters to change a tape drive's operation. Otherwise, it will remain at its last setting.

The tape system characteristics that **settape** automatically resets may only be modified with **mkgpib** or **ioctl**. If you use **ioctl**, the characteristics are modified directly, without affecting /etc/gpib/dev.

If **settape** reports that invalid options exist in the /etc/gpib/dev file, you must reset the options with **mkgpib**.

setuser -- set up a user account for use with Keyshell

SYNTAX
setuser

EXPLANATION
The System Manager uses the **setuser** command to set up or modify a user account for use with the Keyshell user interface, assemblers, compilers, and integration tools.

Specifically, **setuser** sets up the **IU** and **uP** shell variables and configures a Keyshell script for you when you log in. (This script is placed in the user's home directory under .ksh.)

NOTE

> Access to assemblers, compilers, and integration tools is not available within Keyshell until the user's account has been set up with **setuser**.

The **setuser** command prompts you (the System Manager) to supply information about the user's hardware configuration, and about software products that will be used.

sh -- invoke the TNIX shell (command interpreter and programming language)

EXAMPLES

sh -x commandfile	Starts a new shell, then echoes and executes commands in commandfile. When the command file is finished, the new shell goes away.
sh	Invokes a new interactive shell. Entering a CNTRL-D character returns you to your previous shell.

SYNTAX

sh [**-c** string] [**-eiknstuvx**] command-list ...

PARAMETERS

If no parameters are specified, commands will be read from standard input.

-c string	Read input from string.
-e	Exit immediately if a command fails (only applies if shell is not interactive).
-i	Interactive shell. This is the default if input and output are attached to a terminal. See below for details.
-k	Place all keyword arguments in the environment for a command, as opposed to only those keyword arguments that precede the command.
-n	Read commands but do not execute them.
-s	Read commands from standard input. Shell output is written to file descriptor 2.
-t	Exit after reading and executing one command.
-u	Treat unset variables as an error, if a command tries to substitute an unset variable.
-v	Echo shell input lines as they are read.
-x	Print commands and their arguments as they are executed.
-	Turn off the **-x** and **-v** parameters.

command-list
: A list of TNIX or shell commands to execute.

EXPLANATION
The **sh** command is both a programming language and a command interpreter that executes TNIX commands. For information on how to use the shell programming language, refer to the **Shell Programming** section of your **8560 Series System Users Manual.**

Shell Programming Commands

```
if            while       until         for          case in
  then        do          do            do           esac
  else        done        done          done
  elif
   ...
fi

break         continue    eval          exit         export
read          readonly    set           shift        trap
umask
```

Built-in Shell Commands

```
exec          logout      login         times        wait
```

Table 1-8 lists the commonly used shell metacharacters.

FILES
/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO
The **Shell Programming** section of the **8560 Series MUSDU System Users Manual.**

exit

EXIT STATUS
Errors detected by the shell, such as syntax errors cause the shell to return a non zero exit status. If the shell is being used non interactively then execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also **exit**).

Table 1-8
Shell Metacharacters

Syntax	Description
|	Pipe symbol--connects the output of one command to the input of another command
&&	Execute following command if preceding command returns true, that is, sets the $? exit status variable to zero
||	Execute following command if preceding command returns false, that is, sets the $? exit status variable to zero
;	Command separator--separates commands typed on the same line
;;	case delimiter--terminates a case label and associated command list in a case statement statement
&	Background commands--executes the preceding command or parenthesis-enclosed command list concurrently
(...)	Command grouping--parenthesis-enclosed commands are executed in a separate shell process
<	Redirect input from a file or file descriptor
<<	Redirect input from the following text
>	Redirect output to a file or file descriptor
>>	Redirect output, appending to an existing file
*	Match any character, including the null character, other than the slash "/" character
?	Match any single character
[...]	Match any one of the enclosed characters or range of characters
${...}	Substitute shell variable
`...`	Substitute command output
\	Overrides the shell's interpretation of the next character
'...'	Overrides the shell's interpretation of the enclosed characters, except for the apostrophe (') character
"..."	Overrides the shell's interpretation of the enclosed characters except for the quotation (") mark, dollar sign ($), accent grave (`), and backslash (\) characters

shutdown -- bring the 8560 system down in an orderly manner

SYNTAX
shutdown

EXPLANATION
The **shutdown** command kills any outstanding user and daemon processes if not more than one user is logged in. After killing all processes (except for the scheduler and initialization processes), **shutdown** executes the **sync** command and writes the output to the console terminal, telling the operator that the system may be turned off.

If there is more than one user logged in, **shutdown** lists who is logged in, and asks you to try again later. If the user running **shutdown** is not a superuser, **shutdown** emits an error message to that effect.

SEE ALSO
sync

NOTES
To use the **shutdown** command, you must be logged in as root.

sleep -- suspend execution for an interval

EXAMPLES

(sleep 3000; a.out)&	Execute program a.out after waiting 3000 seconds. This command line is placed into background so you can continue your work.

SYNTAX
sleep time [; command]

EXPLANATION
The **sleep** command suspends execution of command for time seconds. It is used to execute a command after a certain amount of time as in:

```
(sleep 105 ;date)&
```

In this case, the command line is placed into background and in 105 seconds, the **date** command will be executed.

Another interesting application would be to execute a command every so often, as in:

```
while true
do
    command
    sleep 37
done
```

For example, suppose you want to know when a file called testresult is created. The following shell procedure checks every minute to see if testresult exists:

```
while test ! -f testresult
do
  sleep 60
done
echo 'testresult has arrived! ^G'
```

When testresult has been created, the message 'testresult has arrived' and a bell (^G) are sent to the terminal.

SEE ALSO
For information on the **alarm** system call, see **alarm** in Section 2 of this manual.

For information on the **sleep** subroutine, see **sleep** in Section 3 of this manual.

NOTES
The time argument must be less than 65536 seconds.

slp -- set line printer characteristics

EXAMPLES

slp /dev/lp1 -tabs -nl	Replaces tabs with spaces and replaces newlines with the characters <CR> and <LF> when sending output to lineprinter 1.
slp /dev/lp2 tabs nl='\000\014'	Replaces newlines with the two characters, <NUL> and <FF>, when sending output to lineprinter 2. (The printer will handle tabs.)

SYNTAX

```
                          [ nl          ]
                  [ tabs] [-nl          ]
slp /dev/printer [-tabs] [ nl='string']
```

PARAMETERS

printer The special file for the desired printer device (e.g. /dev/lp1).

tabs Causes the printer to process tabs.

-tabs Process tabs -- Causes each tab to be replaced with the appropriate number of spaces before sending output to printer.

nl Causes the printer to process newlines.

-nl Process newlines -- Causes the newline character (ASCII 012) to be replaced with the characters <CR> and <LF> (ASCII 015 and 012, respectively) before sending output to printer.

nl='string**'**
On output, replace the newline character with string. See the explanation of this command for further details.

EXPLANATION

The **slp** command modifies the characteristics of the software driver for the specified printer. Once you know your printer characteristics, add the proper **slp** command line to the /etc/rc file. The printer then assumes those characteristics each time the system is booted.

When specifying newline processing, the string is copied character for character. You may also enter octal digits corresponding to ASCII characters by preceding each octal number by a backslash. For example, **nl='\000\014'** replaces newline with a null followed by a form feed. Only the first seven characters of string are used. A zero-length string is illegal.

All tab and newline parameters must follow the printer parameter.

SEE ALSO
lp1r
lp2r
stty

NOTES
Because each tab or newline option is processed, independently, all options preceding a bad option have been processed even though the command "failed".

The **slp** command only sets characteristics for printers connected to the lineprinter ports. Use the **stty** command for printers connected to IOP ports.

EXIT STATUS
The **slp** command returns the following exit status codes:

0 The command executed with no problems.

1 A problem occurred.

2 The **slp** command was invoked improperly,

sort -- sort or merge files

EXAMPLES

sort -u +0f +0 list

Prints, in alphabetical order, all the unique spellings in a list of words. Capitalized words differ from lowercase.

sort -t: +2n /etc/passwd

Prints the password file sorted by the UID (user identification) number. This is the third colon-separated field. See **passwd** in Section 5 of this manual.

sort -um +0 -1 dates

Prints the first instance of each month in an already sorted file of (month, day) entries. The **-um** parameters, with just one input file, make the choice of a unique representative from a set of equal lines predictable.

SYNTAX

sort [**-mubdfinrc**] [**-t** c] [+pos1 [-pos2]]...
[**-o** file] [**-T** directory] [file...]

PARAMETERS

-b Ignores leading blanks (spaces and tabs) in field comparisons.

-d "Dictionary" order: only letters, digits and blanks are significant in comparisons.

-f Fold uppercase letters onto lowercase.

-i Ignore characters outside the ASCII range 040-0176 in non-numeric comparisons.

-n An initial numeric string, consisting of optional blanks, an optional minus sign, and zero or more digits with an optional decimal point, is sorted by arithmetic value. The **-n** parameter implies the **-b** parameter.

-r Reverse the sense of comparisons.

-t c "Tab character" separating fields is c.

The notation +pos1 and -pos2 restricts a sort key to a field beginning at pos1 and ending just before pos2. pos1 and pos2 each have the form m.n, and may be followed by one or more of the **bdfinr** parameters, where m tells a number of fields to skip from the beginning of the line, and n tells a

number of characters to skip further. If any flags are present they override all the global ordering options for this key. If the -b parameter is in effect, n is counted from the first non-blank in the field. The parameter -b is attached independently to pos2. A missing .n means .0; a missing - pos2 means the end of the line. Under the -t c parameter, fields are strings separated by c; otherwise, fields are non-empty non-blank strings separated by blanks.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

The following parameters also apply:

-c Checks that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

-m Merges only, the input files are already sorted.

-o file The next argument is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs.

-T directory
The next argument is the name of a directory in which temporary files should be made.

-u Suppresses all but one in each set of equal lines. Ignored bytes and bytes outside keys do not participate in this comparison.

EXPLANATION

The **sort** command sorts lines of all the named files together and writes the result on the standard output. Specifying "-" as file causes standard input to be used as input. If no input files are named, standard input is sorted.

The default sort key is an entire line. Default ordering is lexicographic by bytes.

FILES

/usr/tmp/stm*

/tmp/*: First and second tries for temporary files.

SEE ALSO
For information on the **passwd** file format, see **passwd** in Section 5 of this manual.

comm
uniq

EXIT STATUS

0	No errors were encountered.
non-zero	Comments and exits with non-zero status for various trouble conditions, including disorder discovered when using the -c parameter.

NOTES
Very long lines are silently truncated.

statgpib -- display GPIB status information

SYNTAX
statgpib [device...]

PARAMETERS
device Specifies the complete pathname of a GPIB-compatible device. The pathname must always begin with /dev/. You may specify several devices with one **statgpib** command.

EXPLANATION
The **statgpib** command displays the status of GPIB-compatible devices on the system. If you specify a device, **statgpib** displays the following information about the device:

- The device's name.
- The device's GD (device driver routine) name.
- The device's primary address (and secondary address, if appropriate) on the GPIB.
- The device's current state. (For example, the device may be busy or exclusively opened.)

If you don't specify a device, **statgpib** displays the following information about the 8560 itself:

- The 8560's primary GPIB address.
- Whether or not the 8560 is currently the GPIB controller-in-charge.
- Whether or not the 8560 is the GPIB system controller.
- Which device (if any) has exclusive use of the GPIB.

EXAMPLES

NOTE

The displays shown here are only examples. The actual displays you receive may differ.

Let's assume that you want to know the current GPIB state of the 8560. To get this information, you type:

```
statgpib

GPIB address = 0.
controller in charge
system controller
no device assigned
```

Now assume that your system includes a GPIB-compatible tape drive called "mt1". This command gives you the tape drive's status:

```
statgpib /dev/mt1

device = /dev/mt1; GD-name = magtape
GPIB address = 4.
state = closed, exclusive use
```

If there are several different tape drives on your system, and you only want the status of mt0 and mt1, you can type:

```
statgpib /dev/mt0 /dev/mt1

device = /dev/mt0; GD-name = magtape
GPIB address = 5.
state = closed, exclusive use

device = /dev/mt1; GD-name = magtape
GPIB address = 4.
state = closed, exclusive use
```

Or, to get the status of all tape drives on your system, you can type:

```
statgpib /dev/mt*

device = /dev/mt0; GD-name = magtape
GPIB address = 5.
state = closed, exclusive use

device = /dev/mt1; GD-name = magtape
GPIB address = 4.
state = closed, exclusive use

device = /dev/mt2; GD-name = magtape
GPIB address = 6.
state = open, exclusive use, busy

device = /dev/mt3; GD-name = magtape
GPIB address = 4.
state = closed, exclusive use
```

FILES

/etc/gpib/gd Contains each GD name, and an associated GPIB Interface identification number.

/etc/gpib/dev Contains the definitions of all GPIB-compatible devices on the system.

SEE ALSO
8560 Magnetic Tape Interface Users Manual

For information on the **gpib** file, see **gpib** in Section 4 of this manual.

For information on the **gpib** file format, see **gpib** in Section 5 of this manual.

ldgpib
mkgpib

NOTES
Don't try to execute **statgpib** if your system doesn't have a GPIB Interface installed. **If you do, the system may** crash.

stty -- set terminal options

EXAMPLES

stty all Prints the current definition of each attribute that can be set with the **stty** command.

SYNTAX
stty [option...]

PARAMETERS
An appended "i" refers to input, an "o" refers to output. An appended "{io}" represents "eveni" and "eveno"

option Any of the following parameters.

-f Forces the specified change to occur even if the terminal characteristic table has been changed. The characteristic table is a larger data structure associated with the terminal. This table can be set through using the **ioctl** system call to change more terminal characteristics than can be changed with **stty**. Without this parameter, you receive a warning message if the characteristic table has been changed and you attempt to set an option. The option is not set when this warning is given.

even{io} Even parity on input, output.

odd{io} Odd parity on input, output.

nopar{io} No parity (bit 7 is always 0) on input, output.

mark{io} Mark parity (bit 7 is always 1) on input, output.

data{io} Data on input, output (8th bit is not stripped).

nocare{io} Don't care parity on input, output.

raw Raw mode input (no erase, kill, interrupt, quit, logout; parity bit passed back).

-raw Negates raw mode.

cooked Negates raw mode.

cbreak Makes each character available to the **read** system call as received; no erase and kill.

-cbreak Makes characters available to **read** only when newline is received.

-nl Allows carriage return for newline and outputs **CR-LF** for carriage return or newline.

nl Accepts only newline (linefeed) to end lines.

echo Echoes back every character typed.

-echo Do not echo characters.

aresm Turns on auto resume, which causes any input to resume the output if it has been suspended.

-aresm Turns off auto resume.

dtr The 8560 monitors pin 20 and controls the DTR line. The 8560 stops output when the DTR line goes low and resumes output when the DTR line goes high. The command:

```
$ stty dtr<CR>
```

allows you to turn off your terminal without getting logged out by the 8560.

-dtr The 8560 logs you out if the DTR line goes low.

xonxof Turns on XON/XOF (CNTL-Q/CNTL-S) flagging on input.

-xonxof Turns off XON/XOF (CNTL-Q/CNTL-S) flagging on input.

cts The 8560 monitors pin 5 and controls the CTS line. The 8560 sets the CTS line low when it cannot accept any more input.

-cts The 8560 keeps the CTS line high.

-tabs Replaces tabs by a proportionate number of spaces when printing.

tabs Preserves tabs.

ek Resets erase and kill characters back to normal ^H and ^U.

erase c Sets erase character to c; c can be of the form "^X" which is interpreted as a CONTROL X.

kill c Sets kill character to c.

300 600 1200 2400 4800 9600
Sets terminal baud rate to the number given, if possible.

IU Specifies that an 8540 or 8550 Integration Unit (IU) is connected to the port. Support for an IU must be enabled before any commands can be issued to the IU or before any control characters used by the HSI protocol can be sent to a terminal connected to the IU. It is recommended that IU support be turned on during system startup before any traffic occurs on the port.

-IU Turns off Integration Unit (IU) support.

EXPLANATION

The **stty** command sets certain I/O options on the current output terminal. With no argument, it reports the current option settings.

SEE ALSO

For information on the **ioctl** system call, see **ioctl** in Section 2 of this manual.

ioctl

ERROR MESSAGES

Invalid option This warning tells you that the configuration table has been changed. See **ioctl** in Section 2 of this manual.

su -- switch user temporarily

EXAMPLES

```
$ su georgem
Password:          [TNIX prompts you for the password here]
```

If the correct password is given, TNIX substitutes georgem for the current user.

SYNTAX

su [username]

PARAMETERS

username A login name. If not specified, defaults to "root".

EXPLANATION

The **su** command demands the password of the specified username. If it is correct, it changes to the specified username and invokes the shell without changing the current directory or the user environment. (See **environ** in Section 5 of this manual.) Using **su** is similar to logging in. (See **login** in Sections 1 and 3 of this manual.)

The new username stays in force until the shell exits (using the **logout** command, for example), at which time control returns to the previous user.

If you specify no username, "root" (the superuser) is assumed. To remind the superuser of his responsibilities, the shell substitutes the pound sign (#) for its usual prompt.

SEE ALSO

For information on the **environ** file format, see **environ** in Section 5 of this manual.

For information on the **login** file format, see **login** in Sections 1 and 3 of this manual.

login
sh

sync -- execute the **sync** system call

SYNTAX
sync

EXPLANATION
The **sync** command executes the **sync** system call. If the system is to be stopped, **sync** must be called to be sure that all disk write operations have been completed before the processor is halted.

SEE ALSO
For information on the **sync** system call, see **sync** in Section 2 of this manual.

update

syschk -- check and optionally repair file system integrity

SYNTAX
syschk [**-bmw**] [**-t** tempdir] [filesystem...]

PARAMETERS

-b Scans for and removes bad blocks from the device. Typing the **-b** parameter is the same as typing both the **-b** and **-m** parameters together. The **syschk** command's default is to not scan for bad blocks. As many bad blocks as possible (up to one per track) are removed by formatting the bad track(s). The **syschk** command's formatting method preserves any good data that was on the track prior to formatting. Any bad blocks that remain after formatting are allocated to the bad block inode (i-number 1) and deallocated from their original structure.

-m Allows modifications (repairs) to be performed on the file system. The **syschk** command's default is not to repair any faults it finds in the file system.

The filesystem being repaired must be **umounted** before you use the **-m** or **-b** parameters.

-w Prints warnings. The **syschk** command's default is not to warn of unusual file system conditions that are not necessarily errors.

-t tempdir
Uses tempdir as a directory for temp files, rather than the default /tmp. This parameter is ignored by the standalone version.

EXPLANATION
There are two **syschk** programs, a standalone version and a command version. The command version is invoked as shown here. See **standalone** in Section 1 of this manual for information on how to run standalone programs.

For each filesystem that you specify, **syschk** performs a number of checks on the integrity of that filesystem. If you do not specify a filesystem, the root filesystem is checked. The standalone version of **syschk** checks the root filesystem (the filesystem beginning on fixed disk 0). The command version filesystem defaults are listed in the file **/etc/checklist**. If **/etc/checklist** cannot be read by **syschk**, **/dev/rhd0** will be used as the default.

The command version will check any filesystem, or repair any filesystem that is not currently mounted. The standalone version is provided to repair the root filesystem, which is always mounted when TNIX is running.

FILES

lost+found	The directory into which orphan files (allocated files that have no links to them) are linked. A lost+found directory should be created under the root directory of each filesystem.
/etc/passwd	The password file.
/etc/group	Used to convert user and group IDs to names.
/etc/checklist	The file of default file systems to check.
/tmp/sysc????	Temp file used by the command version of **syschk**. The standalone version uses for its temp file the last 1001 blocks of the disk on which the root filesystem (the one beginning on fixed disk 0) ends.

SEE ALSO
8560 Series System Managers Guide

NOTES
The **syschk** command runs in either of two environments:

- As a standalone program running to check or repair the root file system, **syschk** uses the swap space (which must reside on the last disk drive occupied by the root filesystem) as temporary storage.
- As a user command running to check the root file system or repair some other file system, **syschk** creates a file under /tmp to store its intermediate data.

Because of the heuristic nature of file system repair, **syschk** can only make reasonable guesses concerning the cause of a set of file system corruptions. The files that **syschk** offers to remove should be examined first to avoid unnecessary file deletion.

Any errors described in this document as "fatal" are severe enough that further checking of the filesystem is pointless; **syschk** cannot repair such a filesystem. The filesystem must be completely restored from the most recent backup, using **restor**.

Since the i-list is a fixed array on the disk, bad blocks in the i-list cannot be deallocated from it.

Intermittent read errors during checking will give confusing results.

A single corruption may result in several errors, all of which will disappear when the corruption is repaired. For example, a block allocation problem involving a directory may cause many link count errors that disappear when the block allocation problem is corrected.

EXIT STATUS

The command version of **syschk** exits with a status of zero (0) if no problems occurred, **one** if some file system corruption was detected, or other values if some other problem occurred (such as invocation of **syschk** with an illegal parameter). The standalone version completes after displaying the message:

syschk complete. halt or reboot.

ERROR MESSAGES

The following is a list of **syschk**'s responses to file system corruptions. When a message ends in a question mark, you can choose not to perform the given repair by typing an "n" or <CR>. Any message regarding a particular inode (file) is preceded by the name of the file (as well as can be determined), the size and mode of the file, the names or IDs of the file's owner, and the times associated with the file. (Times are given in GMT by the standalone **syschk**, and in local time by the command version.) Any message regarding a corrupt link is preceded by the same information about the directory involved and the name of the link.

'filename' **IS NOT A BLOCK OR CHARACTER DEVICE check regardless?**

The filesystem that **syschk** was told to check is not a device. The wrong name was probably typed when **syschk** was invoked. The **syschk** command is offering to check the file anyway.

TOO MANY BAD BLOCKS ignoring bad block 'block'

There are more bad blocks on the disk than **syschk** can record. The given bad block is not being recorded as bad. This message indicates a serious problem with the disk. Perhaps the disk is not ready for I/O, or the disk controller is at fault.

'count' **NEW BAD BLOCKS add to bad block inode?**

There are blocks on the disk that have become bad since the last addition to the bad block inode. The **syschk** command is offering to allocate the new bad blocks to the bad block inode, preventing them from being used in the future.

TOO MANY I-NODES FI('count')

The superblock (block 0 on the disk device) indicates that there are more inodes than can be represented in an i-number (more than 65535). This is a fatal error.

TOO FEW I-NODES ('count')

The superblock (block 0 on the disk device) indicates that there are fewer than two inodes on the file system. This is a fatal error.

I-NODE AREA OVERFLOWS FILE SYSTEM

The superblock (block 0 on the disk device) indicates that the entire file system is consumed or overflowed by the inode area. This is a fatal error.

BAD SUPERBLOCK UPDATE TIME 'time' update?

The superblock's (block 0 on the disk device) indication of the last time it was changed doesn't make sense. The **syschk** command is offering to replace this value with the current time. This check is omitted by the standalone version.

ROOT I-NODE IS UNALLOCATED

The inode dedicated to the root directory is not allocated. This is a fatal error.

ROOT I-NODE IS NOT A DIRECTORY

The inode dedicated to the root directory is allocated, but is something other than a directory. This is a fatal error.

BAD BLOCK I-NODE IS NOT A REGULAR FILE put one there?

The inode dedicated to storing bad blocks should be an ordinary file but it is not. The **syschk** command is offering to rewrite that inode with one representing an ordinary file.

CANNOT ACCESS BAD BLOCK I-NODE

The bad block file cannot be opened. One possibility is that the inode itself is in a bad block and cannot be read. The previous record of what blocks were bad is ignored.

BAD MODE destroy?

The given file is allocated, but it is not one of the following types: directory, regular file, character special file, block special file. The **syschk** command is offering to deallocate the file, destroying its contents.

BAD SIZE destroy?

The size of the given file does not make sense; it is negative or greater than what can be represented. The **syschk** command is offering to deallocate the file, destroying its contents.

DIRECTORY MISALIGNED shrink?

The size of the given directory is not a multiple of the size of a directory entry. The **syschk** command is offering to discard the partial entry.

REFERS TO A BAD BLOCK zero pointer?

The given file contains a bad block. The **syschk** command is offering to change the file so that it refers to a block of zeros instead of the bad block.

(warning) has 'count' fewer blocks than are required

The given file has "holes" in it -- blocks that have never been written to. No action is taken. Because a file can legitimately have such holes, a warning is produced only if the **-w** parameter was specified when **syschk** was invoked.

NO '.' ENTRY add one?

The given directory has no entry to refer to itself. The **syschk** command is offering to build one.

NOTE

Some programs assume that the "." and ".." entries are the first two entries in each directory. A directory repaired in this way may no longer fit that assumption, and may not be treated properly by such programs.

HAS NO '..' ENTRY make one?

The given directory has no entry to refer to its parent directory. The **syschk** command is offering to build one. (See the Note under "**NO '.' ENTRY**".)

I-NUMBER 'number' OUT OF RANGE remove entry?

The given directory entry refers to a nonexistent inode. The **syschk** command is offering to remove the offending link.

ASSOCIATED I-NODE IS UNALLOCATED remove entry?

The given directory entry refers to a deleted file. The **syschk** command is offering to remove the offending link.

'.' ENTRY IS INCORRECT fix?

The "." directory entry does not refer to this directory. The **syschk** command is offering to replace the offending link with a correct one.

'..' ENTRY IS INCORRECT fix?

The ".." directory entry does not refer to this directory's parent. The **syschk** command is offering to replace the offending link with a correct one.

IS AN ORPHAN connect to /lost+found?

The given file, although allocated, has no directory entries referring to it. The **syschk** command is offering to create a link to this file in the lost and found directory (the directory lost+found on the filesystem being repaired). The link created will have the form "Innn" where nnn is the i-number of the orphan file.

CANNOT EXTEND DIRECTORY TO MAKE 'filename'

The given filename cannot be put in the directory. Either the directory cannot be read reliably, or no free blocks are available for allocation. A subsequent message offers an alternative.

NO /lost+found DIRECTORY

A link cannot be made in the lost and found directory because there is no such directory on the filesystem. Manual intervention is required if you want to recover orphan files. Here is the procedure:

1. Using **syschk**, fix all problems on the filesystem except any dealing with orphan files.

2. Running TNIX normally, create the directory lost+found on the appropriate filesystem. For example, the lost+found directory on the filesystem mounted as /usr1 would be called usr1/lost+found.

3. Run **syschk** again, this time attempting to recover orphan files.

CONNECTION FAILED destroy orphan?

Either the orphan file cannot be linked to, or you typed "n" in response to the "connect to lost+found" message. The **syschk** command is offering to delete the file.

INCORRECT LINK COUNT record='count'**, actual='count' fix?**

The inode record of the number of directory entries referring to it is not correct. The **syschk** command is offering to correct the inode's record.

HAS 'count' **BLOCKS OUT OF BOUNDS,** 'count' **DUPLICATED BLOCKS consider removal?**

The given file contains blocks that are also members of other structures. The **syschk** command is offering to remove the file (in terms of total number of block allocation problems on the disk). If you type "y", **syschk** will eventually display the message:

> **If that inode is removed, total out-of bounds will be "**count**" total duplicates will be "**count**" remove it?**

or:

If that inode is removed, no block allocation problems will remain here. remove it?

In either case, typing "y" will delete the file. A file should only be deleted if the deletion would reduce the number of block allocation problems on the disk. The initial "consider removal" message prints the current totals; the later "remove it" message prints the resulting totals.

BAD COUNT IN FREELIST BLOCK rebuild free list?

The free list is corrupt. More precisely, the "number of blocks pointed to by this block" field in a free list block is negative or greater than the maximum allowed. The **syschk** command is offering to replace the free list with a good one.

FREE BLOCK POINTER OUT OF RANGE rebuild free list?

The free list is corrupt. More precisely, one of the block pointers in the free list refers to something other than a data block. The **syschk** command is offering to replace the free list with a good one.

FREELIST CONTAINS A BAD BLOCK rebuild free list?

One of the blocks in the free list cannot be read reliably. The **syschk** command is offering to replace the free list with a good one.

FREELIST DUPLICATES A BLOCK rebuild free list?

The free list is corrupt. More precisely, one block is a member of the free list more than once, or is a member of the free list and some other structure (e.g., a file). The **syschk** command is offering to replace the free list with a good one.

BLOCKS ARE MISSING rebuild free list?

After all filesystem structures have been examined, some blocks remain unaccounted for. The **syschk** command is offering to replace the free list with a good one.

free list is ok. rebuild free list?

The **syschk** command is offering to replace the good free list with a sorted one. Although this action is not necessary to repair a filesystem, sorting the free list may improve performance slightly.

changes were made to the filesystem. check again?

This message indicates that **syschk** has corrected some filesystem problems and that the filesystem should be checked again. A "n" response to this question will avoid the check.

tail -- display the last or first part of a file

EXAMPLES

tail -c -e 300 cmp.c Displays the last 300 **characters** of file cmp.c.

tail -l -s 10 tstme Displays the first 10 **lines** of file tstme.

SYNTAX

```
     [-b]
     [-c]
tail [-l]  [-r]  [[-s number] [-e number]]  [infile]
```

PARAMETERS

-	See infile.
-b	Indicates that number is in units of 512-byte blocks.
-c	Indicates that number is in units of characters.
-l	Indicates that number is in units of lines. This is the default for **number** if **-b**, **-c**, or **-l** is unspecified.
-r	Specifies lines to be printed in reverse order.
-s number	Specifies that the **first** n blocks/characters/lines of file infile be printed (where n is specified by number).
-e number	Specifies that the **last** n blocks/characters/lines of file infile be printed (where n is specified by number). If you specify this option with the **-l** parameter, you must be sure that number is less than 4097 characters. The units that number is in default to lines (see **-l**) if unspecified.
number	Represents how many 512-byte blocks, characters, or lines to print. If the **-b**, **-c**, or **-l** parameter is not specified, the default is **-l** (lines).
infile	The pathname of the input file. If you specify "-" here, standard input is used until a <CTRL-D> character (signaling an end-of-file) is entered.

EXPLANATION

The **tail** command writes part of the named file to standard output beginning at the designated line. If you do not specify infile, **tail** reads from standard input.

SEE ALSO
tty

NOTES
The **tail** command may not work properly with character special files such as /dev/tty (see **tty**).

tee -- send output to both standard output and an optional file

EXAMPLES

cat fname | tee fnamecopy

The output from the command **cat fname** is piped to the input of the **tee** command. The **tee** command sends a copy of its input (unchanged) to standard output (your terminal) and to the file fnamecopy.

SYNTAX
tee [-ia] [file ...]

PARAMETERS

-i Causes interrupts to be ignored.

-a Causes the output to be **appended** to file rather than overwriting it.

EXPLANATION
The **tee** command sends an identical copy of standard input to standard output **and** to any specified file(s).

test -- evaluate a boolean expression

EXAMPLES

```
test "$TERM" ! = "unicorn" && echo not using a unicorn terminal!
```

Displays "not using a unicorn terminal" if the value of the TERM shell variable is not equal to "unicorn".

```
test -d /usr/jonb
```

Returns 0 (true).

```
test -n
```

Returns 0 (true).

```
test 100 -gt 1000
```

Returns 1 (false) because 1000 is greater than 100.

SYNTAX

test expression

PARAMETERS

expression An expression to evaluate.

EXPLANATION

The **test** command evaluates the expression expression, then returns a zero exit status (the value of the "$?" shell variable) if the value of the expression is true. A non-zero exit status is returned if the value of expression is false. The **test** command returns a non-zero exit if you do not specify an expression for **test** to evaluate.

You can use **test** in much the same manner as you would a **conditional** in an if/then/else control structure. The following primitives are used to construct the expression.

-r file True if the file exists and is readable.

-w file True if the file exists and is writable.

-f file True if the file exists and is not a directory.

-d file True if the file exists and is a directory.

-s file True if the file exists and has a size greater than zero.

-t number True if the open file whose file descriptor number is number (1 by default) is associated with a terminal device.

-z string True if the length of string is zero.

-n string True if the length of string is non-zero.

string1 = string2
True if the the values of string1 and string2 are equal.

string1 != string2
True if the the values of string1 and string2 are not equal.

string True if string is not the null string.

n1 **-eq** n2 True if the integers n1 and n2 are algebraically equal. Any of the comparisons **-ne**, **-gt**, **-ge**, **-lt**, or **-le** may be used in place of **-eq**.

These primitives may be combined with the following operators:

! Unary negation operator.

-a Binary and operator.

-o Binary or operator.

(expr) Parentheses for grouping.

Order of Precedence

-a (and) has a higher precedence than **-o**.

NOTE

All parameters are separate arguments to test. Also, parentheses are interpreted by the Shell, so they must be enclosed with quotes, for example, ')'.

SEE ALSO
sh
find

time -- time a command

EXAMPLES

time ps -a Causes the **ps -a** command to be executed and then prints out the time characteristics.

SYNTAX
time command

PARAMETERS

command A TNIX command name.

EXPLANATION
The given command is executed. After it is complete, **time** prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

The times are printed on the diagnostic output stream.

NOTES
Elapsed time is accurate to the second, while the CPU times are measured to the inverse of the line frequency. Thus, the sum of the CPU times can be up to a second longer than the elapsed time.

touch -- update the last modified date of a file

EXAMPLES

touch * Causes all files in the current directory to be updated.

touch -c lastcopy
Causes file lastcopy to be updated.

SYNTAX

touch [-c] **file...**

PARAMETERS

-c Don't create a file.

EXPLANATION

The **touch** command attempts to set the modified date of each file. This is done by reading a character from the file and writing it back.

If a file does not exist, an attempt will be made to create it unless the **-c** parameter is specified.

tr -- translate characters

EXAMPLES

```
tr '+' '\012' < src.file > dest.file
```

Convert all "+" characters in the input file src.file to newline (octal 12) characters and place the result into the output file dest.file.

SYNTAX

tr [-cds] [string1 [string2]

PARAMETERS

-c Complements the set of characters in string1 with respect to the universe of characters whose ASCII codes are 01--0377 (octal).

-d Deletes all input characters in string1.

-s Squeezes all strings of repeated output characters that are in string2 to single characters.

string1 A string of alphanumeric characters to be searched for and replaced with the string of characters specified in string2.

string2 A string of alphanumeric characters to replace any occurrence of a string of characters specified by string1.

EXPLANATION

The **tr** command copies the standard input to the standard output, substituting or deleting selected characters. Input characters found in string1 are mapped into the corresponding characters of string2. When string2 is shorter than string1, it is padded to the length of string1 by duplicating the last character in string2. Any combination of the parameters **-cds** may be used.

In either string, the notation a-b means a range of characters from a to b in increasing ASCII order. The character "\" followed by 1--3 octal digits stands for the character whose ASCII code is given by those digits. A "\" followed by any other character stands for that character.

The following example creates a list in file2 of all the words in file1 (one per line), where a word is taken to be a maximal string of alphabetics. The second string is quoted to protect "\" from the Shell. The ASCII code for newline is 012.

```
tr -cs A-Za-z '\012' <file1 >file2
```

SEE ALSO
ed

NOTES
The **tr** command will not handle the ASCII NUL character in string1 or string2. Instead, it always deletes NUL from input.

true, false -- set the shell exit status variable

EXAMPLES

true Sets the shell exit status variable, "$?", to 0.

false Sets the shell exit status variable, "$?", to 1.

SYNTAX

true
false

EXPLANATION

The **true** command sets the shell exit status variable, "$?", to 0. The **false** command sets the shell exit status variable, "$?" to 1. The **true** and **false** commands are typically used in a shell program to continually repeat a command, for example:

```
while true
do
        echo "Time to go home!^G"
        sleep 60
done
```

To see how **true** and **false** work, type:

```
$ true; echo $?
$ false; echo $?
```

SEE ALSO

sh

EXIT STATUS

0 Stands for "true".

1 Stands for "false".

tset -- initialize terminals

EXAMPLES

tset 4105 Sets terminal-dependent parameters for use with the Tektronix 4105 terminal.

SYNTAX
tset [**-s**] [**-Q**] [type]

PARAMETERS

-Q Prevents **tset** from printing the messages "Erase set to..." and "Kill set to...".

-s Causes **tset** to issue **export** and assignment commands that place information about your terminal from /etc/termcap into your TERMCAP environment variable.

type Your terminal type (4105, for example).

EXPLANATION
The **tset** command causes terminal-dependent processing, such as setting erase and kill characters, setting or resetting delays. It is driven by the /etc/termcap file.

The terminal type is specified by the type argument. The type may be any type given in /etc/termcap. If type is not specified, the terminal type is read from the TERM environment variable.

If any **type** begins with a question mark, you will be asked if you really want that type. A null response means to use that type; otherwise, another type can be entered which will be used instead. (The question mark must be escaped to prevent filename expansion by the shell.)

The erase character is changed to a Control-H (backspace) on terminals that can backspace but not overstrike (such as a CRT), and when the erase character is the default erase character ("#" on standard systems).

The **-s** parameter outputs "export" and assignment commands. It can be used as:

```
tset -s ... > /tmp/tset$$
. /tmp/tset$$
rm /tmp/tset$$
```

The **tset** command is most useful when included in the **.profile** file executed automatically at login.

FILES

/etc/termcap Terminal capability database.

SEE ALSO

For information on the **termcap** file format, see **termcap** in Section 5 of this manual.

tty -- display your terminal's logical device name

EXAMPLES

tty Display the logical device name of the terminal at which you are working.

EXPLANATION

The **tty** command displays the logical device name of the terminal at which you logged in. Each system device (such as terminals and disk drives) has a unique logical device name.

Each device on the system requires an entry name in the **\dev** directory. For a complete list of all device names type:

```
ls -a /dev
```

FILES

/dev The directory that lists all device names on your system.

ERROR MESSAGES

not a tty The standard input is not connected to a terminal. By default, standard input, standard output, and standard error files are initially associated with your login terminal.

uload -- upload/download unformatted data

EXAMPLES

```
COMM              [Type on the 8002 to connect to TNIX.]
                  [Login to TNIX at this point.]
uload -u -s 2 | lpr <CTRL-@><8002-file    [<CTRL-@> is the ASCII NUL
                                           character.]
                  [File transfer takes place now.]
*RIOT* EOJ        [Printed by the 8002.]
<CTRL-D>          [Type in the TNIX end-of-file indicator.]
                  [The file will now by printed by the TNIX lineprinter.]
```

Sends (uploads) an 8002 file (8002-file) to be printed by **TNIX**.

SYNTAX

```
      [-d]
uload [-u] [-p prompt-string] [-s nn] file
```

PARAMETERS

-d Specifies download from the TNIX file file to the 8540/8550/8001/8002. **This is the default parameter.**

-u Specifies upload from the 8540/8550/8001/8002 to the TNIX file file.

-p prompt-string
An optional prompt. This corresponds to the P parameter in the TEKDOS **COMM** command. On **uploads, it is used to prompt for each line (defaults to an empty string). On downloads, it is output to** indicate that the download is finished (defaults to '$$$$').

-s nn Slows down the file transfer by padding each line with nn null characters (**<CTRL-@>**). A slower transfer may be required by TEKDOS on the 8002.

file The pathname of the file to transfer.

EXPLANATION

The **uload** command provides the host side protocol for unformatted text transfers to or from **TNIX**. This is the protocol used for unformatted upload and download by the 8550 system. file contains (or will contain) the unformatted text. On uploading, file defaults to standard input. On downloading, file defaults to standard output.

SEE ALSO

mload
stty

NOTES

Prompt sequences (**-p** parameter) that include dollar signs (**$**) must be enclosed within apostrophes (**'**). For instance, to specify the prompt **$$**, you would enter **-p '$$'** within the **uload** command line.

The echo mode (set with the **stty** command) is disabled during the transfer. The interrupt signal, **<CTRL-C>**, aborts the transfer.

uniq -- report repeated lines in a file

EXAMPLES

uniq -d testfile >dups Causes a listing of all duplicate lines from file testfile to be output to file dups.

SYNTAX
uniq [**-udc** [**+n**] [**-n**]] [input [output]]

PARAMETERS

-u Causes a copy of each line that does **not** have one or more duplicates immediately following itself to be output. For example, if two or more lines in a file were identical and adjacent (each one immediately following the other), a copy of one of these lines would **not** be output. But, if two lines that were identical were separated one from another by one or more different lines (both preceding and following them), they would both be output.

-d Causes **one** copy of a line that has one or more duplicates immediately following itself to be output. Any line that does not have an adjacent duplicate is **not** output. For example, if two or more lines in a file were identical and adjacent (each one immediately following the other), **one** copy of these lines would be output.

-c Generates an output report in default style, but with each line preceded by a count of the number of times that the line appeared adjacently. This parameter supercedes the **-u** and **-d** parameters.

+n The first n characters are ignored. Fields are skipped before characters.

-n The first n fields, together with any blanks before each field are ignored. A field is defined as a string of non-space, non-tab characters separated from its neighbors by tabs and spaces.

input The pathname of the input file.

output The pathname of the output file.

EXPLANATION
The **uniq** command reads file input and compares adjacent lines. By default (when no parameters are specified except for input or output), any second and succeeding copies of duplicated lines are removed. Thus, the output is simply the input file with any adjacent identical lines removed.

SEE ALSO
comm
sort

update -- periodically execute the **sync** system call

SYNTAX
/etc/update

EXPLANATION
The **update** program executes the **sync** system call every 30 seconds. This insures that the file system is fairly up to date in case of a crash. This command should not be executed directly, but should be executed out of the /etc/rc system initialization command file.

SEE ALSO
For information on the **sync** system call, see **sync** in Section 2 of this manual.

init
sync

NOTES
With **update** running, a file system can be damaged if the CPU is halted just as **sync** is executed.

wait -- await completion of process

EXAMPLES

```
nroff report > report.fmt &
wait
```

Waits until the backup job (**nroff**) is finished before reporting any abnormal terminations.

SYNTAX

wait

EXPLANATION

Wait until all processes started with **&** have completed, and report on abnormal terminations.

Because the **wait** system call must be executed in the parent process, the shell executes **wait** directly, without creating a new process.

SEE ALSO

For information on the **wait** system call, see **wait** in Section 2 of this manual.

sh

NOTES

Not all the processes of a 3-or-more-stage pipeline are children of the Shell, and thus can't be waited for.

wall -- write to all users

EXAMPLES

```
# /etc/wall

        All Users:  System going down in 5 minutes.

                    Please log off immediately!

<CTRL-D>
```

This procedure informs all users that the system is being shut down.

SYNTAX

/etc/wall

EXPLANATION

The **wall** command reads its standard input until an end-of-file (a line with only a control-D character). It then sends this message, preceded by

'Broadcast Message ...'

to all logged in users.

To use the **wall** command, you must be logged in as root, in order to override any protections the users may have invoked.

FILES

/dev/tty?
/etc/utmp

SEE ALSO

mesg
write

ERROR MESSAGES

Cannot send to 'user'

The **wall** command cannot write to 'user'. The receiver's tty file is not open. The **mesg n** command closes your tty file so that other users cannot use **wall**, or any other method, to write to your login terminal.

wc -- count lines, words, and characters in a file.

EXAMPLES

wc prog.pas	Counts the lines, words, and characters in the file prog.pas.
who | wc -l	Reports how many people are logged in.

SYNTAX
wc [-lwc] [file...]

PARAMETERS

-l	lines -- Counts the number of lines in a file.
-w	words -- Counts the number of words in a file.
-c	characters -- Counts the number of characters in a file.
file	The pathname of a file.

EXPLANATION
The **wc** command counts lines, words and characters in the named files. If you do not specify filename, **wc** counts the lines, words, and characters received from standard input.

A word is a string of characters delimited by spaces, tabs or newlines.

who -- find out who is on the system

EXAMPLES

who	Shows the login name, terminal name, and login time for all users currently logged in.
whoami	Shows your login name, terminal name, and login time.
who are you?	Shows the login name, terminal name, and login time for the person logged into the terminal at which you typed the command.

SYNTAX
who [file]

who string1 string2

PARAMETERS

file	The pathname of a file.
string	An arbitrary word (usually some recognizable alphanumeric string of letters).

EXPLANATION
Without an argument, **who** lists the login name, terminal name, login date, and login time for each user who is currently logged in.

Without an argument, **who** examines the /etc/utmp file to obtain its information. If a file is given, that file is examined. Typically the given file will be /usr/adm/wtmp, which contains a record of all the logins since it was created. Then **who** lists logins, logouts, and crashes since the creation of the wtmp file. Each login is listed with user name, terminal name (with '/dev/' suppressed), date, and time. When an argument is given, logouts produce a similar line without a user name. Reboots produce a line with 'x' in place of the device name, and a fossil time indicative of when the system went down.

With the second syntax form, any two parameters may be supplied. The **who** command then displays information, such as the login name associated with the terminal you are using.

Suppose you were going to use a terminal and noticed that someone was already logged in and had left. You could use the **who** command to find out who is logged in at a terminal:

```
$ who am i
lorii   ttyi2   Apr 22 08:03
```

FILES
/etc/utmp
/usr/adm/wtmp

SEE ALSO
For information on the **getuid** system call, see **getuid** in Section 2 of this manual.

write -- send a message to another user's terminal

EXAMPLES

write lorii Communicate with user lorii.

SYNTAX
write user [ttyname]

PARAMETERS

user The login name of the user you want to communicate with.

ttyname The pathname of the terminal (device) at which that user is logged in.

EXPLANATION
The **write** command copies lines from your terminal to that of another user. When first called, it sends the message

Message from yourname yourttyname...

The recipient of the message should write back at this point. Communication continues until an end-of-file (CTRL-D) is read from the terminal, or an interrupt (CTRL-C) is sent. At that point **write** sends an end-of-file (EOF) signal to the other terminal and exits.

If you want to write to a user who is logged in more than once, the ttyname argument may be used to indicate the appropriate terminal name.

Permission to write may be denied or granted by use of the **mesg** command. Certain commands, such as **pr**, disallow messages to prevent messy output.

If the character "!" is found at the beginning of a line, **write** calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using **write**: when you first write to another user, wait for him to write back before starting to send. Each party should end each message with a signal for the other user to reply--(o) for "over" is conventional. When conversation is about to be terminated, (oo) is suggested for "over and out".

FILES
/etc/utmp Used to find a user.
/bin/sh Used to execute "!".

SEE ALSO
mesg
who
mail

Section 2

SYSTEM CALLS

INTRODUCTION

This section contains an explanation of the system calls that you may use in your C programs or assembly language programs. (The C compiler and the PDP-11 assembler are included only as part of the optional 8560 Series Native Programming Package.) System calls allow your program to obtain operating system services from the TNIX kernel. The system calls are arranged alphabetically. An index of system calls follows this introduction.

The "Error Messages" topic in this section explains the system call error messages returned by the **perror** standard subroutine.

NOTATION

You will see an integer in parentheses after some of the command names. This indicates that the command is described in the specified section of this manual. For example, **perror(3)** indicates that the **perror** standard subroutine is described in Section 3 of this manual.

ASSEMBLER

Some of the system call descriptions have a topic called "ASSEMBLER". This topic explains factors to be considered when using the system call in a PDP-11 assembly language program.

To assemble a PDP-11 assembly language program that includes system reference calls, use the following command:

as /usr/include/sys.s file...

(The command **as** is available in the optional 8560 Series Native Programming Package.)

Assembly Language Considerations

Return values from system calls appear in registers r0 and r1; the contents of these registers may not be preserved when no return value is expected. An erroneous call is always indicated by turning on the c-bit of the condition codes. The error number is returned in r0. The presence of an error is most easily tested by the instructions bes and bec (branch on error set (or clear)). (These are synonyms for the bcs and bcc instructions.)

ERROR MESSAGES

Most system calls return an error status to the calling process, allowing the process to monitor the status of the system call. For most of the calls, a returned value of -1 indicates an error. Additionally, an error number is available to the calling process in the external variable **errno**. **Errno** can be passed to the standard subroutine **perror**(3), which returns a terse error message to the standard error file. To use **perror** the line **#include <errno.h>** must be included in your source code. **Errno** is not cleared on successful calls, so it should be tested only after an error has occurred.

Here is a list of the error numbers, their names as defined in <errno.h> and as printed to standard error by **perror**, and the meaning of the messages.

0 Error 0

Not used.

1 EPERM Not owner

Typically this error indicates an attempt to modify a file in some way forbidden except to its owner or super-user. It is also returned for attempts by ordinary users to do things allowed only to the super-user.

2 ENOENT No such file or directory

This error occurs when a file name is specified and the file should exist but doesn't, or when one of the directories in a path name does not exist.

3 ESRCH No such process

The process whose number was given to **signal** and **ptrace** does not exist, or is already dead.

4 EINTR Interrupted system call

An asynchronous signal (such as interrupt or quit), which the user has elected to catch, occurred during a system call. If execution is resumed after processing the signal, it will appear as if the interrupted system call returned this error condition.

5 EIO I/O error

Some physical I/O error occurred during a **read** or **write**. This error may in some cases occur on a call following the one to which it actually applies.

6 ENXIO No such device or address

I/O on a special file refers to a subdevice that does not exist, or beyond the limits of the device.

7 E2BIG Arg list too long

An argument list longer than 5120 bytes is presented to **exec.**

8 ENOEXEC Exec format error

A request is made to execute a file which, although it has the appropriate permissions, does not start with a valid magic number. See **a.out(5).**

9 EBADF Bad file number

Either a file descriptor refers to no open file, or a read (write) request is made to a file that is open only for writing (reading).

10 ECHILD No children

Wait and the process has no living or unwaited-for children.

11 EAGAIN No more processes

In a **fork,** the system's process table is full or the user is not allowed to create any more processes.

12 ENOMEM Not enough core

During an **exec** or **break,** a program asks for more core than the system is able to supply. This is not a temporary condition; the maximum core size is a system parameter. The error may also occur if the arrangement of text, data, and stack segments requires too many segmentation registers.

13 EACCES Permission denied

An attempt was made to access a file in a way forbidden by the protection system.

14 EFAULT Bad address

The system encountered a hardware fault in attempting to access the arguments of a system call.

15 ENOTBLK Block device required

A plain file was mentioned where a block device was required, for example, in **mount.**

16 EBUSY Mount device busy

An attempt was made to mount a device that was already mounted or an attempt was made to dismount a device on which there is an active file (open file, current directory, mounted-on file, active text segment).

17 EEXIST File exists

An existing file was mentioned in an inappropriate context, e.g. **link**.

18 EXDEV Cross-device link

A link to a file on another device was attempted.

19 ENODEV No such device

An attempt was made to apply an inappropriate system call to a device; e.g. read a write-only device.

20 ENOTDIR Not a directory

A non-directory was specified where a directory is required, for example in a path name or as an argument to **chdir**.

21 EISDIR Is a directory

An attempt was made to write on a directory.

22 EINVAL Invalid argument

Some invalid argument: dismounting a non-mounted device, mentioning an unknown signal in **signal**, reading or writing a file for which **seek** has generated a negative pointer. Also set by math functions.

23 ENFILE File table overflow

The system's table of open files is full, and temporarily no more **opens** can be accepted.

24 EMFILE Too many open files

The current process has opened too many files. Some must be closed before another can be opened.

25 ENOTTY Not a typewriter

The file mentioned in **stty** or **gtty** is not a terminal or one of the other devices to which these calls apply.

26 ETXTBSY Text file busy

An attempt to execute a pure-procedure program that is currently open for writing (or reading). Also, an attempt to open for writing a pure-procedure program that is being executed.

27 EFBIG File too large

The size of a file exceeded the maximum (about 1.0E9 bytes).

28 ENOSPC No space left on device

During a **write** to an ordinary file, there is no free space left on the device.

29 ESPIPE Illegal seek

An **lseek** was issued to a pipe. This error should also be issued for other non-seekable devices.

30 EROFS Read-only file system

An attempt to modify a file or directory was made on a device mounted read-only.

31 EMLINK Too many links

An attempt to make more than 32767 links to a file.

32 EPIPE Broken pipe

A write on a pipe for which there is no process to read the data. This condition normally generates a signal; the error is returned if the signal is ignored.

33 EDOM Math argument

The argument of a function in the math package (M) is out of the domain of the function.

34 ERANGE Result too large

The value of a function in the math package (M) cannot be represented within machine precision.

SYSTEM CALL INDEX

File Access

access accessibility of file -- 2-9
chmod change mode of file -- 2-13
chown change owner and group of a file -- 2-14
fstat see **stat**
getgid see **getuid**
getuid get user and group identity -- 2-24
setgid see **setuid**
setuid set user and group ID -- 2-43
stat get file status -- 2-46
umask set file creation mode mask -- 2-52
utime set file times -- 2-54

File Reading and Writing

close close a file -- 2-15
creat create a new file -- 2-16
gtty see **ioctl**
ioctl control I/O device -- 2-26
lseek move read/write pointer -- 2-31
open open for reading or writing -- 2-35
read read from file -- 2-42
stty see **ioctl**
tell see **lseek**
write write on a file -- 2-56

Program Execution

break see **brk**

brk change memory allocation -- 2-11

environ see **exec**

exec execute a file -- 2-18

exit terminate process -- 2-21

fork spawn new process -- 2-22

getpid get process identification number -- 2-23

indir indirect system call -- 2-25

kill send signal to a process -- 2-28

nice set program priority -- 2-34

pause stop until signal -- 2-36

phys allow a process to access physical addresses -- 2-37

sbrk see **brk**

ptrace process trace -- 2-40

signal catch or ignore signals -- 2-44

wait wait for process to terminate -- 2-55

System Services

alarm schedule signal after specified time -- 2-10

ftime see **time**

kill send signal to a process -- 2-28

stime set time -- 2-49

time get date and time -- 2-50

File System

chdir change default directory -- 2-12

chroot see **chdir**

dup duplicate an open file descriptor -- 2-17

dup2 see **dup**

link link to a file -- 2-29

mknod make a directory or a special file -- 2-32

mount mount or remove file system -- 2-33

pipe create an interprocess channel -- 2-38

stat get file status -- 2-46

umount see **mount**

unlink remove directory entry -- 2-53

Memory Management

brk change memory allocation -- 2-11

lock lock a process in primary memory -- 2-29

Kernel

profil execution time profile -- 2-39

times get process times -- 2-51

sync update super-block -- 2-49

access - determine accessibility of file

SYNTAX
access(name, mode)
char *name;

EXPLANATION
Access checks the given file name for accessibility according to mode, which is 4 (read), 2 (write) or 1 (execute) or a combination thereof. Specifying mode 0 tests whether the directories leading to the file can be searched and the file exists.

An appropriate error indication is returned if name cannot be found or if any of the desired access modes would not be granted. On disallowed accesses -1 is returned and the error code is in errno, 0 is returned from successful tests.

The user and group IDs with respect to which permission is checked are the real UID and GID of the process, so this call is useful to set-UID programs.

Notice that only access bits are checked. A directory may be announced as writable by access, but an attempt to open it for writing will fail (although files may be created there); a file may look executable, but exec will fail unless it is in proper format.

SEE ALSO
stat(2)

ASSEMBLER
(access = 33.)
sys access; name; mode

alarm - schedule signal after specified time

SYNTAX
alarm(seconds)
unsigned seconds;

EXPLANATION
Alarm causes signal SIGALRM to be sent to the invoking process in a number of seconds given by the argument. Unless caught or ignored, the signal terminates the process. (See **signal(2).**)

Alarm requests are not stacked; successive calls reset the alarm clock. If the argument is 0, any alarm request is canceled. Because the clock has a 1-second resolution, the signal may occur up to one second early; because of scheduling delays, resumption of execution when the signal is caught may be delayed an arbitrary amount. The longest specifiable delay time is 65535 seconds.

The return value is the amount of time previously remaining in the alarm clock.

SEE ALSO
pause(2), signal(2)

ASSEMBLER
(alarm = 27.)
(seconds in r0)
sys alarm
(previous amount in r0)

brk, sbrk, break - change memory allocation

SYNTAX
char *brk(addr)

char *sbrk(incr)

EXPLANATION
Brk sets the system's idea of the lowest location not used by the program (called the break) to addr (rounded up to the next multiple of 64 bytes). Locations not less than addr and below the stack pointer are not in the address space and will thus cause a memory violation if accessed.

In the alternate function sbrk, incr more bytes are added to the program's data space and a pointer to the start of the new area is returned.

When a program begins execution via exec the break is set at the highest location defined by the program and data storage areas. Ordinarily, therefore, only programs with growing data areas need to use break.

SEE ALSO
exec(2)

DIAGNOSTICS
Zero is returned if the break could be set; -1 if the program requests more memory than the system limit or if too many segmentation registers would be required to implement the break.

Sbrk returns a pointer to the start of the new area if successful, and 0 if not.

NOTES
Setting the break in the range 0177701 to 0177777 is the same as setting it to zero.

ASSEMBLER
(break = 17.)
sys break; addr

Break performs the function of brk. The name of the routine differs from that in C for historical reasons.

chdir, chroot - change default directory

SYNTAX
```
chdir(dirname)
char *dirname;

chroot(dirname)
char *dirname;
```

EXPLANATION
Dirname is the address of the pathname of a directory, terminated by a null byte. Chdir causes this directory to become the current working directory, the starting point for path names not beginning with '/'.

Chroot sets the root directory, the starting point for path names beginning with '/'. The call is restricted to the super-user.

Chdir and chroot only change the working and root directories for the current process and any of its children.

SEE ALSO
cd(1)

DIAGNOSTICS
Zero is returned if the directory is changed; -1 is returned if the given name is not that of a directory or is not searchable.

ASSEMBLER
```
(chdir = 12.)
sys chdir; dirname

(chroot = 61.)
sys chroot; dirname
```

chmod - change mode of file

SYNTAX
chmod(name, mode)
char *name;

EXPLANATION
The file whose name is given as the null-terminated string pointed to by name has its mode changed to mode. Modes are constructed by ORing together some combination of the following octal values:

```
04000 set user ID on execution
02000 set group ID on execution
01000 save text image after execution
00400 read by owner
00200 write by owner
00100 execute (search on directory) by owner
00070 read, write, execute (search) by group
00007 read, write, execute (search) by others
```

If an executable file is set up for sharing then mode 01000 prevents the system from abandoning the swap-space image of the program-text portion of the file when its last user terminates. Thus when the next user of the file executes it, the text need not be read from the file system but can simply be swapped in, saving time. Ability to set this bit is restricted to the super-user since swap space is consumed by the images; it is only worthwhile for heavily used commands.

Only the owner of a file (or the super-user) may change the mode. Only the super-user can set the 1000 mode.

SEE ALSO
chmod(1)

DIAGNOSTICS
Zero is returned if the mode is changed; -1 is returned if name cannot be found or if current user is neither the owner of the file nor the super-user.

ASSEMBLER
(chmod = 15.)
sys chmod; name; mode

chown - change owner and group of a file

SYNTAX
```
chown(name, owner, group)
char *name;
```

EXPLANATION
The file whose name is given by the null-terminated string pointed to by name has its owner and group changed as specified. Only the super-user may execute this call, because if users were able to give files away, they could defeat the (nonexistent) file-space accounting procedures.

SEE ALSO
chown(1), passwd(5)

DIAGNOSTICS
Zero is returned if the owner is changed; -1 is returned on illegal owner changes.

ASSEMBLER
(chown = 16.)
```
sys chown; name; owner; group
```

close - close a file

SYNTAX
close(fildes)

EXPLANATION
Given a file descriptor such as returned from an open, creat, dup or **pipe(2)** call, close closes the associated file. A close of all files is automatic on exit, but since there is a limit on the number of open files per process, close is necessary for programs which deal with many files.

Files are closed upon termination of a process, and device file descriptors may be closed automatically by a successful **exec(2)** , following an **ioctl(2)** call.

SEE ALSO
creat(2), dup(2), open(2), pipe(2), exec(2), ioctl(2)

DIAGNOSTICS
Zero is returned if a file is closed; -1 is returned for an unknown file descriptor.

ASSEMBLER
(close = 6.)
(file descriptor in r0)
sys close

creat - create a new file

SYNTAX
creat(name, mode)
char *name;

EXPLANATION
Creat creates a new file or prepares to rewrite an existing file called name, given as the address of a null-terminated string. If the file did not exist, it is given mode mode, as modified by the process's mode mask (see **umask(2)**). Also see **chmod(2)** for the construction of the mode argument.

If the file did exist, its mode and owner remain unchanged but it is truncated to 0 length.

The file is also opened for writing, and its file descriptor is returned.

The mode given is arbitrary; it need not allow writing. This feature is used by programs which deal with temporary files of fixed names. The creation is done with a mode that forbids writing. Then if a second instance of the program attempts a creat, an error is returned and the program knows that the name is unusable for the moment.

SEE ALSO
write(2), close(2), chmod(2), umask(2)

DIAGNOSTICS
The value -1 is returned if: a needed directory is not searchable; the file does not exist and the directory in which it is to be created is not writable; the file does exist and is unwritable; the file is a directory; there are already too many files open.

ASSEMBLER
(creat = 8.)
sys creat; name; mode
(file descriptor in r0)

dup, dup2 - duplicate an open file descriptor

SYNTAX
dup(fildes)
int fildes;

dup2(fildes, fildes2)
int fildes, fildes2;

EXPLANATION
Given a file descriptor returned from an open, pipe, or creat call, dup allocates another file descriptor synonymous with the original. The new file descriptor is returned.

In the second form of the call, fildes is a file descriptor referring to an open file, and fildes2 is a non-negative integer less than the maximum value allowed for file descriptors. Dup2 causes fildes2 to refer to the same file as fildes. If fildes2 already referred to an open file, it is closed first.

SEE ALSO
creat(2), open(2), close(2), pipe(2)

DIAGNOSTICS
The value -1 is returned if: the given file descriptor is invalid; there are already too many open files.

ASSEMBLER
(dup = 41.)
(file descriptor in r0)
(new file descriptor in r1)
sys dup
(file descriptor in r0)

The dup2 entry is implemented by adding 0100 to fildes.

execl, execv, execle, execve, execlp, execvp, exec, exece, environ - execute a file

SYNTAX

```
execl(name, arg0, arg1, ..., argn, 0)
char *name, *arg0, *arg1, ..., *argn;

execv(name, argv)
char *name, *argv[ ];

execlp(name, arg0, arg1, ..., argn, 0)
char *name, *arg0, *arg1, ..., *argn;

execvp(name, argv)
char *name, *argv[ ];

execle(name, arg0, arg1, ..., argn, 0, envp)
char *name, *arg0, *arg1, ..., *argn, *envp[ ];

execve(name, argv, envp);
char *name, *argv[ ], *envp[ ];

extern char **environ;
```

EXPLANATION

Exec in all its forms overlays the calling process with the named file, then transfers to the entry point of the core image of the file. There can be no return from a successful exec; the calling core image is lost.

Files remain open across exec unless explicit arrangement has been made; see ioctl(2). Ignored signals remain ignored across these calls, but signals that are caught (see signal(2)) are reset to their default values.

Each user has a real user ID and group ID and an effective user ID and group ID. The real ID identifies the person using the system; the effective ID determines his access privileges. Exec changes the effective user and group ID to the owner of the executed file if the file has the 'set-user-ID' or 'set-group-ID' modes. The real user ID and group ID are not affected.

The name argument is a pointer to the null-terminated name of the file to be executed. The pointers arg [0], arg [1] ... address null-terminated strings. Conventionally arg [0] is the name of the file.

From C, two interfaces are available. Execl is useful when a known file with known arguments is being called; the arguments to execl are the character strings constituting the file and the arguments; the first argument is conventionally the same as the file name (or its last component). A 0 argument must end the argument list.

The execv version is useful when the number of arguments is unknown in advance; the arguments to execv are the name of the file to be executed and a vector of strings containing the arguments. The last argument string must be followed by a 0 pointer.

When a C program is executed, it is called as follows:

```
main(argc, argv, envp)
int argc;
char **argv, **envp;
```

where argc is the argument count and argv is an array of character pointers to the arguments themselves. As indicated, argc is conventionally at least one and the first member of the array points to a string containing the name of the file.

Argv is directly usable in another execv because argv [argc] is 0.

Envp is a pointer to an array of strings that constitute the environment of the process. Each string consists of a name, an "=", and a null-terminated value. The array of pointers is terminated by a null pointer. The shell **sh(1)** passes an environment entry for each global shell variable defined when the program is called. See **environ(5)** for some conventionally used names. The C run-time start-off routine places a copy of envp in the global cell environ, which is used by execv and execl to pass the environment to any subprograms executed by the current program. The exec routines use lower-level routines as follows to pass an environment explicitly:

```
execle(file, arg0, arg1, . . . , argn, 0, environ);
execve(file, argv, environ);
```

Execlp and execvp are called with the same arguments as execl and execv, but duplicate the shell's actions in searching for an executable file in a list of directories. The directory list is obtained from the environment.

FILES
/bin/sh shell, invoked if command file found by execlp or execvp

SEE ALSO
fork(2), environ(5)

DIAGNOSTICS
If the return value is -1: the file cannot be found, it is not executable, it does not start with a valid magic number (see **a.out(5)**), maximum memory is exceeded, or the arguments require too much space. An exception to this is with **execvp** and **execlp**, which will execute a shell command file (which does not start with a valid magic number). Even for the super-user, at least one of the execute-permission bits must be set for a file to be executed.

NOTES
If execvp is called to execute a file that turns out to be a shell command file, and if it is impossible to execute the shell, the values of argv[0] and argv[-1] will be modified before return.

```
ASSEMBLER
(exec = 11.)
sys exec; name; argv

(exece = 59.)
sys exece; name; argv; envp
```

When the called file starts execution on the 8560, the stack pointer points to a word containing the number of arguments. Just above this number is a list of pointers to the argument strings, followed by a null pointer, followed by the pointers to the environment strings and then another null pointer. The strings themselves follow; a 0 word is left at the very top of memory.

Addresses increase downward in the following diagram.

```
  sp->          nargs
        arg0
        ...
        argn
        0
        env0
        ...
        envm
        0

 arg0:  <arg0\0>
        ...
 env0:  <env0\0>
        0
```

This arrangement happens to conform well to C calling conventions.

exit - terminate process

SYNTAX
exit(status)
int status;

_exit(status)
int status;

EXPLANATION
Exit is the normal means of terminating a process. Exit closes all the process's files and notifies the parent process if it is executing a wait. The low-order 8 bits of status are available to the parent process.

This call can never return.

The C function **exit** may cause cleanup actions before the final 'sys exit'. The C function **_exit** circumvents all cleanup.

SEE ALSO
wait(2)

ASSEMBLER
(exit = 1.)
(status in r0)
sys exit

fork - spawn new process

SYNTAX
fork( )

EXPLANATION
Fork is the only way new processes are created. The new process's memory image is a copy of that of the caller of fork. The only distinction is the fact that the value returned in the old (parent) process contains the process ID of the new (child) process, while the value returned in the child is 0. Process ID's range from 1 to 30,000. This process ID is used by **wait(2)**.

Files open before the fork are shared, and have a common read-write pointer. In particular, this is the way that standard input and output files are passed and also how pipes are set up.

SEE ALSO
wait(2), **exec(2)**

DIAGNOSTICS
Returns -1 and fails to create a process if: there is inadequate swap space, the user is not super-user and has too many processes, or the system's process table is full. Only the super-user can take the last process-table slot.

ASSEMBLER
(fork = 2.)
sys fork
(new process return)
(old process return, new process ID in r0)

The return locations in the old and new process differ by one word. The C-bit is set in the old process if a new process could not be created.

getpid - get process identification

SYNTAX
getpid( )

EXPLANATION
Getpid returns the process ID of the current process. Most often it is used to generate uniquely-named temporary files.

ASSEMBLER
(getpid = 20.)
sys getpid
(pid in r0)

getuid, getgid, geteuid, getegid - get user and group identity

SYNTAX
getuid( )

geteuid( )

getgid( )

getegid( )

EXPLANATION
Getuid returns the real user ID of the current process, geteuid the effective user ID. The real user ID identifies the person who is logged in, in contradistinction to the effective user ID, which determines his access permission at the moment. It is thus useful to programs which operate using the 'set user ID' mode, to find out who invoked them.

Getgid returns the real group ID, getegid the effective group ID.

SEE ALSO
setuid(2)

ASSEMBLER
(getuid = 24.)
sys getuid
(real user ID in r0, effective user ID in r1)

(getgid = 47.)
sys getgid
(real group ID in r0, effective group ID in r1)

indir - indirect system call

ASSEMBLER
(indir = 0.)
sys indir; call

The system call at the location call is executed. Execution resumes after the indir call.

The main purpose of indir is to allow a program to store arguments for system calls and execute them out of line in the data segment. This preserves the purity of the text segment.

If indir is executed indirectly, it is a no-op. If the instruction at the indirect location is not a system call, indir returns error code EINVAL. Refer to the Error Messages discussion in this section.

ioctl, stty, gtty - control device

SYNTAX

```
#include <tiop.h>

ioctl(fildes, request, argp)
struct sgttyb *argp;

stty(fildes, argp)
struct sgttyb *argp;

gtty(fildes, argp)
struct sgttyb *argp;
```

EXPLANATION

Ioctl performs a variety of functions on character special files (devices).

For certain status setting and status inquiries about terminal devices, the functions stty and gtty are equivalent to

```
ioctl(fildes, TIOCSETP, argp)
ioctl(fildes, TIOCGETP, argp)
```

respectively; see tty(4).

The following two calls apply to any open file:

```
ioctl(fildes, FIOCLEX, NULL);
ioctl(fildes, FIONCLEX, NULL);
```

The first causes the file to be closed automatically during a successful exec operation; the second reverses the effect of the first.

SEE ALSO

stty(1), exec(2), fd(4), hd(4), tty(4)

DIAGNOSTICS

Zero is returned if the call was successful; -1 if the file descriptor does not refer to the kind of file for which it was intended, or if the data structure description is invalid.

NOTES

Strictly speaking, since ioctl may be extended in different ways to devices with different properties, argp should have an open-ended declaration like

```
union { struct sgttyb ...; ... } *argp;
```

The important thing is that the size is fixed by 'struct sgttyb'.

```
ASSEMBLER
(ioctl = 54.)
sys ioctl; fildes; request; argp

(stty = 31.)
(file descriptor in r0)
stty; argp

(gtty = 32.)
(file descriptor in r0)
sys gtty; argp
```

kill - send signal to a process

SYNTAX
kill(pid, sig);

EXPLANATION
Kill sends the signal sig to the process specified by the process number pid. See **signal(2)** for a list of signals.

The sending and receiving processes must have the same effective user ID, otherwise this call is restricted to the super-user.

If the process number is 0, the signal is sent to all other processes in the group of processes originating from the same terminal as the sender.

If the process number is -1, and the user is the super-user, the signal is broadcast universally except to processes 0 and 1, the scheduler and initialization processes. See **init(1)**.

Processes may send signals to themselves.

SEE ALSO
signal(2), kill(1)

DIAGNOSTICS
Zero is returned if the process is killed; -1 is returned if the process does not have the same effective user ID and the user is not super-user, or if the process does not exist.

ASSEMBLER
(kill = 37.)
(process number in r0)
sys kill; sig

link - link to a file

SYNTAX
link(name1, name2)
char *name1, *name2;

EXPLANATION
A link to name1 is created; the link has the name name2. Either name may be an arbitrary path name.

SEE ALSO
ln(1), unlink(2)

DIAGNOSTICS
Zero is returned when a link is made; -1 is returned when name1 cannot be found, when name2 already exists, when the directory of name2 cannot be written, when an attempt is made to link to a directory by a user other than the super-user, when an attempt is made to link to a file on another file system, or when a file has too many links.

ASSEMBLER
(link = 9.)
sys link; name1; name2

lock - lock a process in primary memory

SYNTAX
lock(flag)

EXPLANATION
If the flag argument is non-zero, the process executing this call will not be swapped except if it is required to grow. If the argument is zero, the process is unlocked. This call may only be executed by the super-user.

NOTES
Lock ed processes interfere with the compaction of primary memory and can cause deadlock. This system call is not considered a permanent part of the system.

ASSEMBLER
(lock = 53.)
sys lock; flag

lseek, tell - move read/write pointer

SYNTAX
long lseek(fildes, offset, whence)
long offset;

long tell(fildes)

EXPLANATION
The file descriptor refers to a file open for reading or writing. The read (write) pointer for the file is set as follows:

If whence is 0, the pointer is set to offset bytes.

If whence is 1, the pointer is set to its current location plus offset.

If whence is 2, the pointer is set to the size of the file plus offset.

The returned value is the resulting pointer location.

The function tell (fildes) is identical to lseek (fildes, 0L, 1).

(The 'L' in '0L' specifies a long value in C.)

Seeking far beyond the end of a file, then writing, creates a gap or 'hole', which occupies no physical space and reads as zeros.

SEE ALSO
open(2), creat(2)

DIAGNOSTICS
-1 is returned for an undefined file descriptor, seek on a pipe, or seek to a position before the beginning of file.

NOTES
Lseek is a no-op on character special files.

ASSEMBLER
(lseek = 19.)
(file descriptor in r0)
sys lseek; offset1; offset2; whence

Offset1 and offset2 are the high and low words of offset; r0 and r1 contain the pointer upon return.

mknod - make a directory or a special file

SYNTAX
```
mknod(name, mode, addr)
char *name;
```

EXPLANATION
Mknod creates a new file whose name is the null-terminated string pointed to by name. The mode of the new file (including directory and special file bits) is initialized from mode. (The protection part of the mode is modified by the process's mode mask; see **umask**(2)). The first block pointer of the i-node is initialized from addr. For ordinary files and directories addr is normally zero. In the case of a special file, addr specifies which special file.

Mknod may be invoked only by the super-user.

SEE ALSO
mkdir(1), **filsys**(5)

DIAGNOSTICS
Zero is returned if the file has been made; -1 if the file already exists or if the user is not the super-user.

ASSEMBLER
(mknod = 14.)
```
sys  mknod; name; mode; addr
```

mount, umount - mount or remove file system

SYNTAX
```
mount(special, name, rwflag)
char *special, *name;

umount(special)
char *special;
```

EXPLANATION
Mount announces to the system that a removable file system has been mounted on the block-structured special file special; from now on, references to file name will refer to the root file on the newly mounted file system. Special and name are pointers to null-terminated strings containing the appropriate path names.

Name must exist already. Name must be a directory (unless the root of the mounted file system is not a directory). Its old contents are inaccessible while the file system is mounted.

The rwflag argument determines whether the file system can be written on; if it is 0 writing is allowed, if non-zero no writing is done. Physically write-protected and magnetic tape file systems must be mounted read-only or errors will occur when access times are updated, whether or not any explicit write is attempted.

Umount announces to the system that the special file is no longer to contain a removable file system. The associated file reverts to its ordinary interpretation.

SEE ALSO
mount(1)

DIAGNOSTICS
Mount returns 0 if the action occurred; -1 if special is inaccessible or not an appropriate file, if name does not exist, if special is already mounted, if name is in use, or if there are already too many file systems mounted.

Umount returns 0 if the action occurred; -1 if if the special file is inaccessible or does not have a mounted file system, or if there are active files in the mounted file system.

ASSEMBLER
```
(mount = 21.)
sys  mount; special; name; rwflag

(umount = 22.)
sys  umount; special
```

nice - set program priority

SYNTAX
nice(incr)

EXPLANATION
The scheduling priority of the process is augmented by incr. Positive priorities get less service than normal. Priority 10 is recommended to users who wish to execute long-running programs without inconveniencing other users.

Negative increments are ignored except on behalf of the super-user. The priority is limited to the range -20 (most urgent) to 20 (least).

The priority of a process is passed to a child process by **fork(2)**. For a privileged process to return to normal priority from an unknown state, nice should be called successively with arguments -40 (goes to priority -20 because of truncation), 20 (to get to 0), then 0 (to maintain compatibility with previous versions of this call).

SEE ALSO
nice(1)

ASSEMBLER
(nice = 34.)
(priority in r0)
sys nice

open - open for reading or writing

SYNTAX
open(name, mode)
char *name;

EXPLANATION
Open opens the file name for reading (if mode is 0), writing (if mode is 1) or for both reading and writing (if mode is 2). Name is the address of a string of ASCII characters representing a path name, terminated by a null character.

The file is positioned at the beginning (byte 0). The returned file descriptor must be used for subsequent calls for other input-output functions on the file.

SEE ALSO
creat(2), read(2), write(2), dup(2), close(2)

DIAGNOSTICS
The value -1 is returned if the file does not exist, if one of the necessary directories does not exist or is unreadable, if the file is not readable (writable), or if too many files are open, or if the mode is invalid.

ASSEMBLER
(open = 5.)
sys open; name; mode
(file descriptor in r0)

pause - stop until signal

SYNTAX
pause( )

EXPLANATION
Pause never returns normally. It is used to give up control while waiting for a signal from **kill(2)** or **alarm(2)**.

SEE ALSO
kill(1), kill(2), alarm(2), signal(2)

ASSEMBLER
(pause = 29.)
sys pause

phys - allow a process to access physical addresses

SYNTAX
phys(segreg, size, physadr)

EXPLANATION
The argument segreg specifies a process virtual (data-space) address range of 8K bytes starting at virtual address segreg x8K bytes. This address range is mapped into physical address physadr x64 bytes. Only the first size x64 bytes of this mapping is addressable. If size is zero, any previous mapping of this virtual address range is nullified. For example, the call

```
phys(6, 1, 0177775);
```

will map virtual addresses 0160000-0160077 into physical addresses 017777500-017777577. In particular, virtual address 0160060 is the PDP-11 console located at physical address 017777560.

This call may only be executed by the super-user.

DIAGNOSTICS
The function value zero is returned if the physical mapping is in effect. The value -1 is returned if not super-user, if segreg is not in the range 0-7, if size is not in the range 0-127, or if the specified segreg is already used for other than a previous call to phys.

NOTES
This system call is obviously very machine-dependent and very dangerous. This system call is not considered a permanent part of the system.

ASSEMBLER
(phys = 52.)
sys phys; segreg; size; physadr

pipe - create an interprocess channel

SYNTAX
pipe(fildes)
int fildes[2];

EXPLANATION
The pipe system call creates an I/O mechanism called a pipe. The file descriptors returned can be used in read and write operations. When the pipe is written using the descriptor fildes [1] up to 4096 bytes of data are buffered before the writing process is suspended. A read using the descriptor fildes [0] will pick up the data. Writes with a count of 4096 bytes or less are atomic (i.e. indivisible); no other process can intersperse data.

It is assumed that after the pipe has been set up, two (or more) cooperating processes (created by subsequent fork calls) will pass data through the pipe with read and write calls.

The Shell has a syntax to set up a linear array of processes connected by pipes.

Read calls on an empty pipe (no buffered data) with only one end (all write file descriptors closed) returns an end-of-file.

SEE ALSO
sh(1), read(2), write(2), fork(2)

DIAGNOSTICS
The function value zero is returned if the pipe was created; -1 if too many files are already open. A signal (See **signal**(2).) is generated if a write on a pipe with only one end is attempted.

NOTES
Should more than 4096 bytes be necessary in any pipe among a loop of processes, deadlock will occur.

ASSEMBLER
(pipe = 42.)
sys pipe
(read file descriptor in r0)
(write file descriptor in r1)

profil - execution time profile

SYNTAX

```
profil(buff, bufsiz, offset, scale)
char *buff;
int bufsiz, offset, scale;
```

EXPLANATION

Buff points to an area of memory whose length (in bytes) is given by bufsiz. After this call, the user's program counter (pc) is examined each clock tick (60th or 50th second depending on local line frequency); offset is subtracted from it, and the result multiplied by scale. If the resulting number corresponds to a word inside buff, that word is incremented.

The scale is interpreted as an unsigned, fixed-point fraction with binary point at the left: 0177777(8) gives a 1-1 mapping of pc's to words in buff; 077777(8) maps each pair of instruction words together. 02(8) maps all instructions onto the beginning of buff (producing a non-interrupting memory clock).

Profiling is turned off by giving a scale of 0 or 1. It is rendered ineffective by giving a bufsiz of 0. Profiling is turned off when an exec is executed, but remains on in child and parent both after a fork. Profiling will be turned off if an update in buff would cause a memory fault.

ASSEMBLER

(profil = 44.)

```
sys profil; buff; bufsiz; offset; scale
```

ptrace - process trace

SYNTAX
#include <signal.h>

ptrace(request, pid, addr, data)
int *addr;

EXPLANATION
Ptrace allows a parent process to control the execution of a child process, and examine and change its memory image. Its primary use is for the implementation of breakpoint debugging. There are four arguments whose interpretation depends on a request argument. Generally, pid is the process ID of the traced process, which must be a child (no more distant descendant) of the tracing process. A process being traced behaves normally until it encounters some signal whether internally generated like 'illegal instruction' or externally generated like 'interrupt.' See **signal(2)** for the list. Then the traced process enters a stopped state and its parent is notified via **wait(2)**. When the child is in the stopped state, its memory image can be examined and modified using ptrace. If desired, another ptrace request can then cause the child either to terminate or to continue, possibly ignoring the signal.

The value of the request argument determines the precise action of the call:

0 This request and (under some circumstances) request 10 are the only ones used by the child process. It declares that the process is to be traced by its parent. All the other arguments are ignored. Peculiar results will ensue if the parent does not expect to trace the child.

1 The word in the child process's address space at addr is returned. Addr must be even. The child must be stopped. The input data is ignored.

The word of the system's per-process data area corresponding to addr is returned. Addr must be even and less than 512. This space contains the registers and other information about the process; its layout corresponds to the user structure in the system.

The given data is written at the word in the process's address space corresponding to addr, which must be even. No useful value is returned. Attempts to write in pure text fail if another process is executing the same file.

6 The process's system data is written, as it is read with request 3. Only a few locations can be written in this way: the general registers, the floating point status and registers, and certain bits of the processor status word.

7 The data argument is taken as a signal number and the child's execution continues at location addr as if it had incurred that signal. Normally the signal number will be either 0 to indicate that the signal that caused the stop should be ignored, or that value fetched out of the

process's image indicating which signal caused the stop. If addr is (int *)1 then execution continues from where it stopped.

8 The traced process terminates.

9 Execution continues as in request 7; however, as soon as possible after execution of at least one instruction, execution stops again. The signal number from the stop is SIGTRAP. On the PDP-11 the T-bit is used and just one instruction is executed. This is part of the mechanism for implementing breakpoints.

10 If data is non-zero, then subsequent system calls made below addr are declared illegal and generate a SIGSYS. An addr of 0177777 (on the PDP-11) effectively disables all system calls. If either data or addr is zero, then system calls are again handled normally. This allows system calls to be traced, and is used to simulate trap instructions when emulating DEC environments. Unlike the other requests, pid can refer to the calling process or a traced child. In some emulation schemes, this is the only request used and there is in fact no child.

As indicated, requests 1 through 9 can be used only when the subject process has stopped. The wait call is used to determine when a process stops; in such a case the 'termination' status returned by wait has the value 0177 to indicate stoppage rather than genuine termination.

To forestall possible fraud, ptrace inhibits the set-user-id facility on subsequent **exec(2)** calls. If a traced process calls exec, it will stop before executing the first instruction of the new image showing signal SIGTRAP.

SEE ALSO
wait(2), signal(2)

DIAGNOSTICS
The value -1 is returned if request is invalid, pid is not a traceable process, addr is out of bounds, or data specifies an illegal signal number.

NOTES
The request 0 call should be able to specify signals which are to be treated normally and not cause a stop. In this way, for example, programs with simulated floating point (which use 'illegal instruction' signals at a very high rate) could be efficiently debugged.
The error indication, -1, is a legitimate function value. Errno (refer to the "Introduction" in this section) can be used to determine any errors.

ASSEMBLER
(ptrace = 26.)
(data in r0)
sys ptrace; pid; addr; request
(value in r0)

read - read from file

SYNTAX
read(fildes, buffer, nbytes)
char *buffer;

EXPLANATION
A file descriptor is a word returned from a successful open, creat, dup, or pipe call. Buffer is the location of nbytes contiguous bytes into which the input will be placed. It is not guaranteed that all nbytes bytes will be read; for example if the file refers to a typewriter at most one line will be returned. In any event the number of characters read is returned.

If the returned value is 0, then end-of-file has been reached.

SEE ALSO
open(2), creat(2), dup(2), pipe(2)

DIAGNOSTICS
As mentioned, 0 is returned when the end of the file has been reached. If the read was otherwise unsuccessful the return value is -1. Many conditions can generate an error: physical I/O errors, bad buffer address, preposterous nbytes, file descriptor not that of an input file.

ASSEMBLER
(read = 3.)
(file descriptor in r0)
sys read; buffer; nbytes
(byte count in r0)

setuid, setgid - set user and group ID

SYNTAX
setuid(uid)

setgid(gid)

EXPLANATION
The user ID (group ID) of the current process is set to the argument. Both the effective and the real ID are set. These calls are only permitted to the super-user or if the argument is the real ID.

SEE ALSO
getuid(2)

DIAGNOSTICS
Zero is returned if the user (group) ID is set; -1 is returned otherwise.

ASSEMBLER
(setuid = 23.)
(user ID in r0)
sys setuid

(setgid = 46.)
(group ID in r0)
sys setgid

signal - catch or ignore signals

SYNTAX
#include <signal.h>

(*signal(sig, func))()
(*func)();

EXPLANATION
A signal is generated by some abnormal event, initiated either by user at a typewriter (quit, interrupt), by a program error (bus error, etc.), or by request of another program (kill). Normally all signals cause termination of the receiving process, but a signal call allows them either to be ignored or to cause an interrupt to a specified location. Here is the list of signals with names as in the include file.

```
SIGHUP   1    hangup
SIGINT   2    interrupt
SIGQUIT  3*   quit
SIGILL   4*   illegal instruction (not reset when caught)
SIGTRAP  5*   trace trap (not reset when caught)
SIGIOT   6*   IOT instruction
SIGEMT   7*   EMT instruction
SIGFPE   8*   floating point exception
SIGKILL  9    kill (cannot be caught or ignored)
SIGBUS   10*  bus error
SIGSEGV  11*  segmentation violation
SIGSYS   12*  bad argument to system call
SIGPIPE  13   write on a pipe or link with no one to read it
SIGALRM  14   alarm clock
SIGTERM  15   software termination signal
         16   unassigned
```

The starred signals in the list above cause a memory image if not caught or ignored.

If func is SIG_DFL, the default action for signal sig is reinstated; this default is termination, sometimes with a memory image. If func is SIG_IGN the signal is ignored. Otherwise when the signal occurs func will be called with the signal number as argument. A return from the function will continue the process at the point it was interrupted. Except as indicated, a signal is reset to SIG_DFL after being caught. Thus if it is desired to catch every such signal, the catching routine must issue another signal call.

When a caught signal occurs during certain system calls, the call terminates prematurely. In particular this can occur during a read or **write**(2) on a slow device (like a typewriter, but not a file); and during pause or **wait**(2). When such a signal occurs, the saved user status is arranged in such a way that when return from the signal-catching takes place, it will appear that the system call returned an error status. The user's program may then, if it wishes, re-execute the call.

The value of signal is the previous (or initial) value of func for the particular signal.

After a **fork(2)** the child inherits all signals. **Exec(2)** resets all caught signals to default action.

SEE ALSO
kill(1), kill(2), ptrace(2)

DIAGNOSTICS
The value (int)-1 is returned if the given signal is out of range.

NOTES
If a repeated signal arrives before the last one can be reset, there is no chance to catch it.

The type specification of the routine and its func argument are problematical.

ASSEMBLER
(signal = 48.)
sys signal; sig; label
(old label in r0)

If label is 0, default action is reinstated. If label is odd, the signal is ignored. Any other even label specifies an address in the process where an interrupt is simulated. An RTI or RTT instruction will return from the interrupt.

stat, fstat - get file status

SYNTAX

```
#include <sys/types.h>
#include <sys/stat.h>

stat(name, buf)
char *name;
struct stat *buf;

fstat(fildes, buf)
struct stat *buf;
```

EXPLANATION

Stat obtains detailed information about a named file. Fstat obtains the same information about an open file known by the file descriptor from a successful open, creat, dup or **pipe(2)** call.

Name points to a null-terminated string naming a file; buf is the address of a buffer into which information is placed concerning the file. It is unnecessary to have any permissions at all with respect to the file, but all directories leading to the file must be searchable. The layout of the structure pointed to by buf as defined in <stat.h> is given below. **St_mode** is encoded according to the '#define' statements.

```
struct stat
{
        dev_t   st_dev;         /* device where the inode lives*/
        ino_t   st_ino;         /* inode number */
        unsigned short st_mode; /* mode */
        short   st_nlink;       /* link count   */
        short   st_uid;         /* set user id  */
        short   st_gid;         /* set group id */
        dev_t   st_rdev;        /* device where data lives*/
        off_t   st_size;        /* file size    */
        time_t  st_atime;       /* access time  */
        time_t  st_mtime;       /* modification time    */
        time_t  st_ctime;       /* creation time        */
};


#define S_IFMT          0170000         /* type of file */
#define S_IFDIR         0040000         /* direct */
#define S_IFCHR         0020000         /* character special */
#define S_IFBLK         0060000         /* block special */
#define S_IFREG         0100000         /* regular */
#define S_ISUID         0004000         /* set user id on execution */
#define S_ISGID         0002000         /* set group id on execution */
#define S_ISVTX         0001000         /* save swapped text even */
                                        /* after use    */
#define S_IREAD         0000400         /* read permission, owner */
#define S_IWRITE        0000200         /* write permission, owner */
```

```
#define S_IEXEC         0000100         /* execute/search permission, */
                                        /* owner        */
```

The mode bits 0000070 and 0000007 encode group and others permissions (see **chmod**(2)). The defined types, **ino_t**, **off_t**, **time_t**, name various width integer values; **dev_t** encodes major and minor device numbers; their exact definitions are in the include file <sys/types.h>

When fildes is associated with a pipe, fstat reports an ordinary file with restricted permissions. The size is the number of bytes queued in the pipe.

st_atime is the time that the file was last read. For reasons of efficiency, it is not set when a directory is searched, although this would be more logical. **st_mtime** is the time the file was last written or created. It is not set by changes of owner, group, link count, or mode. **st_ctime** is set both by writing and changing the i-node.

SEE ALSO
ls(1), **filsys**(5)

DIAGNOSTICS
Zero is returned if a status is available; -1 if the file cannot be found.

ASSEMBLER
(stat = 18.)
sys stat; name; buf

(fstat = 28.)
(file descriptor in r0)
sys fstat; buf

stime - set time

SYNTAX

```
#include <sys/types.h>
#include <sys/timeb.h>
stime(tp)
struct timeb *tp;
```

EXPLANATION

stime sets the system's idea of the time and date. Time, pointed to by tp, is a structure defined by <sys/timeb.h>:

```
struct timeb {
        time_t  time;           /* seconds past the epoch*/
        unsigned short millitm; /* up to 1000 milliseconds of*/
                                /* more precise interval*/
        short   timezone;       /* minutes of time westward */
                                /* from Greenwich       */
        short   dstflg;         /* Daylight savings time flag*/
}
```

The epoch is 0000 GMT Jan 1, 1970. Dstflg is a flag which is non-zero if daylight savings time is locally observed at some times during the year. Only the super-user may use this call.

SEE ALSO

date(1), time(2)

DIAGNOSTICS

Zero is returned if the time was set; -1 if user is not the super-user.

ASSEMBLER

(stime = 25.)

sys ftime; tp

sync - update super-block

SYNTAX
sync( )

EXPLANATION
Sync causes all information in core memory that should be on disk to be written out. This includes modified super blocks, modified i-nodes, and delayed block I/O.

It should be used by programs which examine a file system, for example icheck, df, etc. It is mandatory before a boot.

The writing, although scheduled, is not necessarily complete upon return from sync.

SEE ALSO
sync(1), update(1)

ASSEMBLER
(sync = 36.)
sys sync

time, ftime - get date and time

SYNTAX

```
long time(0)

long time(tloc)
long *tloc;

#include <sys/types.h>
#include <sys/timeb.h>
ftime(tp)
struct timeb *tp;
```

EXPLANATION

Time returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds. If tloc is nonnull, the return value is also stored in the place to which tloc points. The ftime entry fills in a structure pointed to by its argument, as defined by <sys/timeb.h> :

```
struct timeb {
      time_t   ttime;
      unsigned short millitm;
      short    timezone;
      short    dstflag;
};
struct timebtz{
      struct timeb t;/* The old timeb structure */
      char     tzname[4];/* time zone name */
      char     tzdayname[4];/* daylight timezone name */
};
```

The timeb structure contains the time since the epoch in seconds, up to 1000 milliseconds of more-precise interval, the local timezone (measured in minutes of time westward from Greenwich), and a flag that, if nonzero, indicates that Daylight Saving time is locally observed at times during the year. The epoch is 0000 GMT Jan. 1, 1970.

If the pointer argument to ftime is an odd address the timebtz structure is returned. It is returned to where the odd address points minus one. The timebtz structure contains the timeb structure plus the standard timezone string, and daylight timezone string.

SEE ALSO

date(1), stime(2)

ASSEMBLER

```
(ftime = 35.)
sys ftime; bufptr

(time = 13.; obsolete call)
sys time
(time since 1970 in r0-r1)
```

times - get process times

SYNTAX
times(buffer)
struct tbuffer *buffer;

EXPLANATION
Times returns time-accounting information for the current process and for the terminated child processes of the current process. All times are in 1/HZ seconds, where HZ= 60 or 50 depending on local line frequency.

After the call, the buffer will appear as follows:

```
struct tbuffer {
        long    proc_user_time;
        long    proc_system_time;
        long    child_user_time;
        long    child_system_time;
};
```

The children times are the sum of the children's process times and their children's times.

SEE ALSO
time(1), time(2)

ASSEMBLER
(times = 43.)
sys times; buffer

umask - set file creation mode mask

SYNTAX
umask(complmode)

EXPLANATION
Umask sets a mask used whenever a file is created by **creat(2)** or **mknod(2)**: the actual mode (see **chmod(2)**) of the newly-created file is the logical **and** of the mode given to chmod, creat, or mknod, and the complement of the argument given to umask. Only the low-order 9 bits of the mask (the protection bits) participate. In other words, the mask shows the bits to be turned off when files are created.

The previous value of the mask is returned by the call. The value is initially 0 (no restrictions). The mask is inherited by child processes.

SEE ALSO
creat(2), mknod(2), chmod(2)

ASSEMBLER
(umask = 60.)
sys umask; complmode

unlink - remove directory entry

SYNTAX
unlink(name)
char *name;

EXPLANATION
Name points to a null-terminated string. Unlink removes the entry for the file pointed to by name from its directory. If this entry was the last link to the file, the contents of the file are freed and the file is destroyed. If, however, the file was open in any process, the actual destruction is delayed until it is closed, even though the directory entry has disappeared.

SEE ALSO
rm(1), link(2)

DIAGNOSTICS
Zero is normally returned; -1 indicates that the file does not exist, that its directory cannot be written, or that the file contains pure procedure text that is currently in use. Write permission is not required on the file itself. It is also illegal to unlink a directory (except for the super-user).

ASSEMBLER
(unlink = 10.)
sys unlink; name

utime - set file times

SYNTAX

```
#include <sys/types.h>
utime(file, timep)
char *file;
time_t timep[2];
```

EXPLANATION

The utime call uses the `accessed' and `updated' times in that order from the timep vector to set the corresponding recorded times for file.

The timep vector is a 2 element array of time_t types.

The caller must be the owner of the file or the super-user. The `inode-changed' time of the file is set to the current time.

SEE ALSO

stat(2)

ASSEMBLER

(utime = 30.)

```
sys utime; file; timep
```

wait - wait for process to terminate

SYNTAX
wait(status)
int *status;

wait(0)

EXPLANATION
Wait causes its caller to delay until a signal is received or one of its child processes terminates. If any child has died since the last wait, return is immediate; if there are no children, return is immediate with an error indication. The normal return yields the process ID of the terminated child. In the case of several children several wait calls are needed to learn of all the deaths.

If (int) status is nonzero, the high byte of the word pointed to receives the low byte of the argument of exit when the child terminated. The low byte receives the termination status of the process. See **signal(2)** for a list of termination statuses (signals); 0 status indicates normal termination. A special status (0177) is returned for a stopped process which has not terminated and can be restarted. See **ptrace(2)**. If the 0200 bit of the termination status is set, a memory image of the process was produced by the system.

If the parent process terminates without waiting on its children, the initialization process (process ID = 1) inherits the children.

SEE ALSO
exit(2), fork(2), signal(2)

DIAGNOSTICS
Returns -1 if there are no children not previously waited for.

ASSEMBLER
(wait = 7.)
sys wait
(process ID in r0)
(status in r1)

The high byte of the status is the low byte of r0 in the child at termination.

write - write on a file

SYNTAX
write(fildes, buffer, nbytes)
char *buffer;

EXPLANATION
A file descriptor is a word returned from a successful open, creat, dup, or pipe(2) call.

Buffer is the address of nbytes contiguous bytes which are written on the output file. The number of characters actually written is returned. It should be regarded as an error if this is not the same as requested.

Writes which are multiples of 512 characters long and begin on a 512-byte boundary in the file are more efficient than any others.

SEE ALSO
creat(2), open(2), pipe(2)

DIAGNOSTICS
Returns -1 on error: bad descriptor, buffer address, or count; physical I/O errors.

ASSEMBLER
(write = 4.)
(file descriptor in r0)
sys write; buffer; nbytes
(byte count in r0)

Section 3

STANDARD SUBROUTINES

INTRODUCTION

The standard subroutines described in this section are a library of programming tools that may be used in your C and assembly language programs. The TNIX system calls, also available for your use, are described in Section 2 of this manual. The subroutines are arranged alphabetically in this section. An index follows this introduction.

NOTATION

You will see an integer in parentheses after some of the command names. This indicates that the command is described in the specified section of this manual. For example, **perror(3)** indicates the **perror** standard subroutine found in Section 3.

SUBROUTINE LIBRARIES

The standard subroutines are located in various libraries. The library is indicated by the letter in parentheses following the subroutine name at the top of the page. The following paragraphs explain which libraries are indicated by these letters.

(C) These functions, together with those of Section 2 (System Calls) and those marked (S), constitute library libc, which is automatically loaded by the C compiler **cc**. The link editor **ld** searches this library under the '-lc' option. (The **cc** and **ld** commands are included Declarations for some of these functions may be obtained from include files indicated on the appropriate pages.

(M) These functions constitute the math library, libm. The link editor searches this library under the '-lm' option. Declarations for these functions may be obtained from the include file <math.h>.

(S) These functions constitute the 'standard I/O package', and are in the library libc. Declarations for these functions may be obtained from the include file <stdio.h>.

(X) Various specialized libraries have not been given distinctive captions. The files in which these libraries are found are named on the appropriate pages.

For more information, refer to the description of the **stdio** subroutine in this section.

ERROR MESSAGES

Functions in the math library (M) may return conventional values when the function is undefined for the given arguments or when the value is not representable. In these cases the external variable errno is set to the value EDOM or ERANGE. (**Errno** is discussed in the Introduction to Section 2 of this manual.) The values of EDOM and ERANGE are defined in the include file <math.h>.

USING SUBROUTINES IN ASSEMBLY LANGUAGE PROGRAMS

In assembly language these subroutines may be accessed by simulating the C calling sequence. For example, **ecvt**(3) might be called this way:

```
setd
mov  $sign,-(sp)
mov  $decpt,-(sp)
mov  ndigit,-(sp)
movf value,-(sp)
jsr  pc,_ecvt
add  $14.,sp
```

SUBROUTINE INDEX

The following is an alphabetical list of each standard subroutine, a short description of its function and the page where it may be found:

abort -- generate IOT fault -- 3-7
abs -- integer absolute value -- 3-8
acos -- see sin
arc -- see plot
asctime -- see ctime
asin -- see sin
assert -- program verification -- 3-9
atan -- see sin
atan2 -- see sin
atof -- convert ASCII to numbers -- 3-10
atoi -- see atof
atol -- see atof
cabs -- see hypot
calloc -- see malloc
ceil -- see floor
circle -- see plot
clearerr -- see feof
closepl -- see plot
cont -- see plot
cos -- see sin
cosh -- see sinh
crypt -- DES encryption -- 3-11
ctime -- convert date and time to ASCII -- 3-12
ctype -- see isalpha
dbm -- see dbmint
dbminit -- data base subroutines -- 3-14
delete -- see dbminit
ecvt -- output conversion -- 3-16
edata -- see end
encrypt -- see crypt
end -- last locations in program -- 3-17
endgrent -- see getgrent
endpwent -- see getpwent
erase -- see plot
etext -- see end
exp -- exponential, logarithm, power, square root -- 3-18
fabs -- see floor
fclose -- close or flush a stream -- 3-19
fcvt -- see ecvt
fdopen -- see fopen
feof -- see feof
ferror -- stream status inquiries -- 3-20
fetch -- see dbminit
fflush -- see fclose
fgetc -- see getc
fgets -- see gets
fileno -- see feof
firstkey -- see dbminit
floor -- absolute value, floor, ceiling functions -- 3-21

fopen -- open a stream -- 3-22
fprint -- see printf
fputc -- see putc
fputs -- see puts
fread -- buffered binary input/output -- 3-23
free -- see malloc
freopen -- see fopen
frexp -- split into mantissa and exponent -- 3-24
fscanf -- see scanf
fseek -- reposition a stream -- 3-25
ftell -- see fseek
fwrite -- see fread
gcd -- see madd
gcvt -- see ecvt
getc -- get character or word from stream -- 3-26
getchar -- see getc
getenv -- value for environment name -- 3-27
getgrent -- get group file entry -- 3-28
getgrgid -- see getgrent
getgrnam -- see getgrent
getlogin -- get login name -- 3-30
getpass -- read a password -- 3-31
getpw -- get name from UID -- 3-32
getpwent -- get password file entry -- 3-33
getpwnam -- see getpwent
getpwuid -- see getpwent
gets -- get a string from a stream -- 3-34
getw -- see getc
hypot -- Euclidean distance -- 3-35
index -- see strcpy
isalnum -- see isalpha
isalpha -- character classification -- 3-36
isascii -- see isalpha
isatty -- see ttyname
iscntrl -- see isalpha
isdigit -- see isalpha
islower -- see isalpha
isprint -- see isalpha
ispunct -- see isalpha
isspace -- see isalpha
isupper -- see isalpha
itom -- see madd
j0, j1, jn -- bessel functions -- 3-37
l3tol -- convert short integers to long integers -- 3-38
label -- see plot
ldexp -- see frexp
line -- see plot
linemod -- see plot
localtime -- see ctime
log -- see exp
log10 -- see exp
longjmp -- see setjmp
ltol3 -- see l3tol
madd -- multiple precision integer arithmetic -- 3-39

malloc -- main memory allocator -- 3-40
mdiv -- see madd
min -- see madd
mktemp -- make a unique file name -- 3-41
modf -- see frexp
monitor -- prepare execution profile -- 3-42
mout -- see madd
move -- see plot
mp -- see madd
msub -- see madd
mult -- see madd
nextkey -- see dbminit
nlist -- get entries from name list -- 3-43
openpl -- see plot
pclose -- see popen
perror -- system error messages -- 3-44
pkopen -- packet driver simulator -- 3-45
plot -- graphics interface -- 3-46
point -- see plot
popen -- initiate I/O to/from a process -- 3-48
pow -- see exp
pow -- see madd
printf -- formated output conversion -- 3-49
putc, putchar -- put character or word on a stream -- 3-51
puts -- put a string on a stream -- 3-52
putw -- see putc
qsort -- quicker sort -- 3-53
rand -- random number generator -- 3-54
realoc -- see malloc
rewind -- see fseek
rindex -- see strcpy
rpow -- see madd
scanf -- formated input conversion -- 3-55
setbuf -- assign buffering to a stream -- 3-58
setgrent -- see getgrent
setjmp -- non-local goto -- 3-59
setkey -- see crypt
setpwent -- see getpwent
sin -- trigonometric functions -- 3-60
sinh -- hyperbolic functions -- 3-61
sleep -- suspend execution for interval -- 3-62
space -- see plot
sprintf -- see printf
sqrt -- see exp
srand -- see rand
sscanf -- see scanf
stdio -- standard buffered input/output package -- 3-63
store -- see dbminit
strcat -- see strcpy
strcmp -- see strcpy
strcpy -- copy string and string operations -- 3-64
strlen -- see strcpy
strncat -- see strcpy
strncmp -- see strcpy

strncpy -- see strcpy
swab -- swap bytes -- 3-65
system -- issue a shell command -- 3-66
tan -- see sin
tanh -- see sinh
timezone -- see ctime
ttyname -- find name of a terminal -- 3-67
ttyslot -- see ttyname
ungetc -- push character back into input stream -- 3-68
y0, y1, yn -- see j0

abort - generate IOT fault

EXPLANATION
Abort executes the PDP11 IOT instruction. This causes a signal that normally terminates the process with a core dump, which may be used for debugging.

SEE ALSO
adb, signal(2), exit(2)
The **adb** command is provided in the optional 8560 Native Programming Package.

DIAGNOSTICS
Usually 'IOT trap - core dumped' from the shell.

abs - integer absolute value

SYNTAX
abs(i)

EXPLANATION
Abs returns the absolute value of its integer operand.

SEE ALSO
floor(3) for information about fabs

NOTES
You get what the hardware gives on the largest negative integer.

assert - program verification

SYNTAX
#include <assert.h>

assert (expression)

EXPLANATION

Assert is a macro that indicates expression is expected to be true at this point in the program. It causes an **exit(2)** with a diagnostic comment on the standard output when expression is false (0). Compiling with the **cc** option **-DNDEBUG** effectively deletes assert from the program. The **cc** command is provided in the optional 8560 Native Programming Package.

DIAGNOSTICS
'Assertion failed: file f line n'. F is the source file and n the source line number of the assert statement.

atof, atoi, atol - convert ASCII to numbers

SYNTAX

```
double atof(nptr)
char *nptr;

atoi(nptr)
char *nptr;

long atol(nptr)
char *nptr;
```

EXPLANATION

These functions convert a string pointed to by nptr to floating, integer, and long integer representation respectively. The first unrecognized character ends the string.

Atof recognizes an optional string of tabs and spaces, then an optional sign, then a string of digits optionally containing a decimal point, then an optional 'e' or 'E' followed by an optionally signed integer.

Atoi and atol recognize an optional string of tabs and spaces, then an optional sign, then a string of digits.

SEE ALSO

scanf(3)

NOTES

There are no provisions for overflow.

crypt, setkey, encrypt - DES encryption

SYNTAX

```
char *crypt(key, salt)
char *key, *salt;

setkey(key)
char *key;

encrypt(block, edflag)
char *block;
```

EXPLANATION

Crypt is the password encryption routine. It is based on the NBS Data Encryption Standard, with variations intended (among other things) to frustrate use of hardware implementations of the DES for key search.

The first argument to crypt is a user's typed password. The second is a 2-character string chosen from the set [a-zA-Z0-9./]. The salt string is used to perturb the DES algorithm in one of 4096 different ways, after which the password is used as the key to encrypt repeatedly a constant string. The returned value points to the encrypted password, in the same alphabet as the salt. The first two characters are the salt itself.

The other entries provide (rather primitive) access to the actual DES algorithm. The argument of setkey is a character array of length 64 containing only the characters with numerical value 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored, leading to a 56-bit key which is set into the machine.

The argument to the encrypt entry is likewise a character array of length 64 containing 0's and 1's. The argument array is modified in place to a similar array representing the bits of the argument after having been subjected to the DES algorithm using the key set by setkey. If edflag is 0, the argument is encrypted; if non-zero, it is decrypted.

SEE ALSO

passwd(1), passwd(5), login(1), getpass(3)

NOTES

The return value points to static data whose content is overwritten by each call.

ctime, localtime, gmtime, asctime, timezone - convert date and time to ASCII

SYNTAX

```
char *ctime(clock)
long *clock;

#include <time.h>

struct tm *localtime(clock)
long *clock;

struct tm *gmtime(clock)
long *clock;

char *asctime(tm)
struct tm *tm;

char *timezone(zone, dst)
```

EXPLANATION

Ctime converts a time pointed to by clock (such as returned by the system call **time**) into ASCII and returns a pointer to a 26-character string in the following form. All the fields have constant width.

```
    Sun Sep 16 01:03:52 1973\n\0
```

Localtime and gmtime return pointers to structures containing the broken-down time. Localtime corrects for the time zone and possible daylight savings time; gmtime converts directly to GMT, which is the time TNIX uses. Asctime converts a broken-down time to ASCII and returns a pointer to a 26-character string.

The structure declaration from the include file is:

```
struct tm { /* see ctime(3) */
        int     tm_sec;
        int     tm_min;
        int     tm_hour;
        int     tm_mday;
        int     tm_mon;
        int     tm_year;
        int     tm_wday;
        int     tm_yday;
        int     tm_isdst;
};
```

These quantities give the time on a 24-hour clock, day of month (1-31), month of year (0-11), day of week (Sunday = 0), year - 1900, day of year (0-365), and a flag that is nonzero if daylight saving time is in effect.

When local time is called for, the program consults the system to determine the time zone and whether the standard U.S.A. daylight saving time adjustment is appropriate. The program knows about the peculiarities of this conversion in 1974 and 1975; if necessary, a table for these years can be extended.

Timezone returns the name of the time zone associated with its first argument, which is measured in minutes westward from Greenwich. If the second argument is 0, the standard name is used, otherwise the Daylight Saving version. If the required name does not appear in a table built into the routine, the difference from GMT is produced; e.g. in Afghanistan timezone(-(60*4+30), 0) is appropriate because it is 4:30 ahead of GMT and the string **GMT+4:30** is produced.

SEE ALSO
time(2)

NOTES
The return values point to static data whose content is overwritten by each call.

dbminit, fetch, store, delete, firstkey, nextkey - data base subroutines

SYNTAX

```
typedef struct { char *dptr; int dsize; } datum;

dbminit(file) char *file;

datum fetch(key) datum key;

store(key, content) datum key, content;

delete(key) datum key;

datum firstkey();

datum nextkey(key); datum key;
```

EXPLANATION

These functions maintain key/content pairs in a data base. The functions will handle very large (a billion blocks) databases and will access a keyed item in one or two file system accesses. The functions are obtained with the loader option **-ldbm**.

Key s and content s are described by the datum typedef. A datum specifies a string of dsize bytes pointed to by dptr. Arbitrary binary data, as well as normal ASCII strings, are allowed. The data base is stored in two files. One file is a directory containing a bit map and has '.dir' as its suffix. The second file contains all data and has '.pag' as its suffix.

Before a database can be accessed, it must be opened by dbminit. At the time of this call, the files file.dir and file.pag must exist. (An empty database is created by creating zero-length '.dir' and '.pag' files.)

Once open, the data stored under a key is accessed by fetch and data is placed under a key by store. A key (and its associated contents) is deleted by delete. A linear pass through all keys in a database may be made, in an (apparently) random order, by use of firstkey and nextkey. Firstkey will return the first key in the database. With any key nextkey will return the next key in the database. This code will traverse the data base:

```
for(key=firstkey(); key.dptr!=NULL; key=nextkey(key))
```

DIAGNOSTICS

All functions that return an int indicate errors with negative values. A zero return indicates ok. Routines that return a datum indicate errors with a null (0) dptr.

NOTES

The '.pag' file will contain holes so that its apparent size is about four times its actual content. Older TNIX systems may create real file blocks for these holes when touched. These files cannot be copied by normal means (cp, cat, tp, tar, ar) without filling in the holes.

Dptr pointers returned by these subroutines point into static storage that is changed by subsequent calls.

The sum of the sizes of a key/content pair must not exceed the internal block size (currently 512 bytes). Moreover all key/content pairs that hash together must fit on a single block. Store will return an error in the event that a disk block fills with inseparable data.

Delete does not physically reclaim file space, although it does make it available for reuse.

The order of keys presented by firstkey and nextkey depends on a hashing function, not on anything interesting.

ecvt, fcvt, gcvt - output conversion

SYNTAX

```
char *ecvt(value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *fcvt(value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *gcvt(value, ndigit, buf)
double value;
char *buf;
```

EXPLANATION

Ecvt converts the value to a null-terminated string of ndigit ASCII digits and returns a pointer thereto. The position of the decimal point relative to the beginning of the string is stored indirectly through decpt (negative means to the left of the returned digits). If the sign of the result is negative, the word pointed to by sign is non-zero, otherwise it is zero. The low-order digit is rounded.

Fcvt is identical to **ecvt**, except that the correct digit has been rounded for Fortran F-format output of the number of digits specified by ndigits.

Gcvt converts the value to a null-terminated ASCII string in buf and returns a pointer to buf. It attempts to produce ndigit significant digits in Fortran F format if possible, otherwise E format, ready for printing. Trailing zeros may be suppressed.

SEE ALSO

printf(3)

NOTES

The return values point to static data whose content is overwritten by each call.

end, etext, edata - last locations in program

SYNTAX
```
extern end;
extern etext;
extern edata;
```

EXPLANATION
These names refer neither to routines nor to locations with interesting contents. The address of etext is the first address above the program text, edata above the initialized data region, and end above the uninitialized data region.

When execution begins, the program break coincides with end, but many functions reset the program break, among them the routines of **brk(2)**, **malloc(3)**, standard input/output (**stdio(3)**), the profile (**-p**) option of **cc**, etc. (The **cc** command is provided in the optional 8560 Native Programming Package.) The current value of the program break is reliably returned by 'sbrk(0)', see **brk(2)**.

SEE ALSO
brk(2), **malloc(3)**

exp, log, log10, pow, sqrt - exponential, logarithm, power, square root

SYNTAX

```
#include <math.h>

double exp(x)
double x;

double log(x)
double x;

double log10(x)
double x;

double pow(x, y)
double x, y;

double sqrt(x)
double x;
```

EXPLANATION

Exp returns the exponential function of x.

Log returns the natural logarithm of x ; log10 returns the base 10 logarithm.

Pow returns x^y.

Sqrt returns the square root of x.

SEE ALSO

hypot(3), sinh(3)

DIAGNOSTICS

Exp and pow return a huge value when the correct value would overflow; errno is set to ERANGE. Pow returns 0 and sets errno to EDOM when the second argument is negative and non-integral and when both arguments are 0.

Log returns 0 when x is zero or negative; errno is set to EDOM.

Sqrt returns 0 when x is negative; errno is set to EDOM.

SEE ALSO

Refer to the Introduction of Section 2 of this manual for information on **errno**.

fclose, fflush - close or flush a stream

SYNTAX
#include <stdio.h>

fclose(stream)
FILE *stream;

fflush(stream)
FILE *stream;

EXPLANATION
Fclose causes any buffers for the named stream to be emptied, and the file to be closed. Buffers allocated by the standard input/output system are freed.

Fclose is performed automatically upon calling **exit(2)**.

Fflush causes any buffered data for the named output stream to be written to that file. The stream remains open.

SEE ALSO
close(2), fopen(3), setbuf(3)

DIAGNOSTICS
These routines return **EOF** if stream is not associated with an output file, or if buffered data cannot be transferred to that file.

feof, ferror, clearerr, fileno - stream status inquiries

SYNTAX

```
#include <stdio.h>

feof(stream)
FILE *stream;

ferror(stream)
FILE *stream

clearerr(stream)
FILE *stream

fileno(stream)
FILE *stream;
```

EXPLANATION

Feof returns non-zero when end of file is read on the named input stream, otherwise zero.

Ferror returns non-zero when an error has occurred reading or writing the named stream, otherwise zero. Unless cleared by clearerr, the error indication lasts until the stream is closed.

Clrerr resets the error indication on the named stream.

Fileno returns the integer file descriptor associated with the stream, see **open(2)**.

These functions are implemented as macros; they cannot be redeclared.

SEE ALSO

fopen(3), open(2)

fabs, floor, ceil - absolute value, floor, ceiling functions

SYNTAX

```
#include <math.h>

double floor(x)
double x;

double ceil(x)
double x;

double fabs(x)
double(x);
```

EXPLANATION

Fabs returns the absolute value |x|.

Floor returns the largest integer not greater than x.

Ceil returns the smallest integer not less than x.

SEE ALSO

abs(3)

fopen, freopen, fdopen - open a stream

SYNTAX
```
#include <stdio.h>

FILE *fopen(filename, type)
char *filename, *type;

FILE *freopen(filename, type, stream)
char *filename, *type;
FILE *stream;

FILE *fdopen(fildes, type)
char *type;
```

EXPLANATION
Fopen opens the file named by filename and associates a stream with it. Fopen returns a pointer to be used to identify the stream in subsequent operations.

Type is a character string having one of the following values:

r open for reading

w create for writing

a append: open for writing at end of file, or create for writing

If a + sign is included in the type string after the type identifier, then the stream will be opened for reading and writing (i.e. mode 2; see **open(2)**).

Freopen substitutes the named file in place of the open stream. It returns the original value of stream. The original stream is closed.

Freopen is typically used to attach the preopened constant names **stdin, stdout, and stderr,** to specified files.

Fdopen associates a stream with a file descriptor obtained from open, dup, creat, or **pipe(2)**. The type of the stream must agree with the mode of the open file.

SEE ALSO
open(2), fclose(3)

DIAGNOSTICS
Fopen and freopen return the pointer **NULL** if filename cannot be accessed.

NOTES
Fdopen is not portable to systems other than TNIX. Types **a+**, **r+**, and **w+** may not be portable. Use with care.

fread, fwrite - buffered binary input/output

SYNTAX

```
#include <stdio.h>

fread(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;

fwrite(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;
```

EXPLANATION

Fread reads, into a block beginning at ptr, nitems of data of the type of *ptr from the named input stream. It returns the number of items actually read.

Fwrite appends at most nitems of data of the type of *ptr beginning at ptr to the named output stream. It returns the number of items actually written.

SEE ALSO

read(2), write(2), fopen(3), getc(3), putc(3), gets(3), puts(3), printf(3), scanf(3)

DIAGNOSTICS

Fread and fwrite return 0 upon end of file or error.

frexp, ldexp, modf - split into mantissa and exponent

SYNTAX

```
double frexp(value, eptr)
double value;
int *eptr;

double ldexp(value, exp)
double value;

double modf(value, iptr)
double value, *iptr;
```

EXPLANATION

Frexp returns the mantissa of a double value as a double quantity, x, of magnitude less than 1 and stores an integer n such that value=x*2**n indirectly through eptr.

Ldexp returns the quantity value*2**exp.

Modf returns the positive fractional part of value and stores the integer part indirectly through iptr.

fseek, ftell, rewind - reposition a stream

SYNTAX

```
#include <stdio.h>

fseek(stream, offset, ptrname)
FILE *stream;
long offset;

long ftell(stream)
FILE *stream;

rewind(stream)
```

EXPLANATION

Fseek sets the position of the next input or output operation on the stream. The new position is at the signed distance offset bytes from the beginning, the current position, or the end of the file, according as ptrname has the value 0, 1, or 2.

Fseek undoes any effects of **ungetc(3)**.

Ftell returns the current value of the offset relative to the beginning of the file associated with the named stream. It is measured in bytes on TNIX.

Rewind(stream) is equivalent to fseek(stream, 0L, 0).

SEE ALSO

lseek(2), fopen(3)

DIAGNOSTICS

Fseek returns -1 for improper seeks.

getc, getchar, fgetc, getw - get character or word from stream

SYNTAX

```
#include <stdio.h>

int getc(stream)
FILE *stream;

int getchar()

int fgetc(stream)
FILE *stream;

int getw(stream)
FILE *stream;
```

EXPLANATION

Getc returns the next character from the named input stream.

Getchar() is identical to getc(stdin).

Fgetc behaves like getc, but is a genuine function, not a macro; it may be used to save object text.

Getw returns the next word from the named input stream. It returns the constant EOF upon end of file or error, but since that is a good integer value, feof and ferror(3) should be used to check the success of getw. Getw assumes no special alignment in the file.

SEE ALSO

fopen(3), putc(3), gets(3), scanf(3), fread(3), ungetc(3)

DIAGNOSTICS

These functions return the integer constant EOF at end of file or upon read error.

A stop with message, 'Reading bad file', means an attempt has been made to read from a stream that has not been opened for reading by fopen.

NOTES

The end-of-file return from getchar is incompatible with that in UNIX editions 1-6.

Because it is implemented as a macro, getc treats a stream argument with side effects incorrectly. In particular, 'getc(*f++);' doesn't work sensibly.

getenv - value for environment name

SYNTAX
char *getenv(name)
char *name;

EXPLANATION
Getenv searches the environment list for a string of the form name = value and returns value if such a string is present, otherwise 0 (NULL).

SEE ALSO
environ(5), exec(2)

getgrent, getgrgid, getgrnam, setgrent, endgrent - get group file entry

SYNTAX

```
#include <grp.h>

struct group *getgrent();

struct group *getgrgid(gid) int gid;

struct group *getgrnam(name) char *name;

int setgrent();

int endgrent();
```

EXPLANATION

Getgrent, getgrgid and getgrnam each return pointers to an object with the following structure containing the broken-out fields of a line in the group file.

```
struct  group { /* see getgrent(3) */
        char    *gr_name;
        char    *gr_passwd;
        int     gr_gid;
        char    **gr_mem;
};
```

The members of this structure are:

gr_name
: The name of the group.

gr_passwd
: The encrypted password of the group.

gr_gid
: The numerical group-ID.

gr_mem
: Null-terminated vector of pointers to the individual member names.

Getgrent simply reads the next line while getgrgid and getgrnam search until a matching gid or name is found (or until EOF is encountered). Each routine picks up where the others leave off so successive calls may be used to search the entire file.

A call to setgrent has the effect of rewinding the group file to allow repeated searches. Endgrent may be called to close the group file when processing is complete.

FILES

/etc/group

SEE ALSO
getlogin(3), getpwent(3), group(5)

DIAGNOSTICS
A null pointer (0) is returned on EOF or error.

NOTES
All information is contained in a static area so it must be copied if it is to be saved.

getlogin - get login name

SYNTAX
char *getlogin();

EXPLANATION
Getlogin returns a pointer to the login name as found in /etc/utmp. It may be used in conjunction with getpwnam to locate the correct password file entry when the same userid is shared by several login names.

If getlogin is called within a process that is not attached to a typewriter, it returns NULL. The correct procedure for determining the login name is to first call getlogin and if it fails, to call getpwuid.

FILES
/etc/utmp

SEE ALSO
getpwent(3), getgrent(3), utmp(5)

DIAGNOSTICS
Returns NULL (0) if name not found.

NOTES
The return values point to static data whose content is overwritten by each call.

getpass - read a password

SYNTAX
char *getpass(prompt)
char *prompt;

EXPLANATION
Getpass reads a password from the file /dev/tty, or if that cannot be opened, from the standard input, after prompting with the null-terminated string prompt and disabling echoing. A pointer is returned to a null-terminated string of at most 8 characters.

FILES
/dev/tty

SEE ALSO
crypt(3)

NOTES
The return value points to static data whose content is overwritten by each call.

getpw - get name from UID

SYNTAX
getpw(uid, buf)
char *buf;

EXPLANATION
Getpw searches the password file for the (numerical) uid, and fills in buf with the corresponding line; it returns non-zero if uid could not be found. The line is null-terminated.

FILES
/etc/passwd

SEE ALSO
getpwent(3), passwd(5)

DIAGNOSTICS
Non-zero return on error.

getpwent, getpwuid, getpwnam, setpwent, endpwent - get password file entry

SYNTAX

```
#include <pwd.h>
struct passwd *getpwent();
struct passwd *getpwuid(uid) int uid;
struct passwd *getpwnam(name) char *name;
int setpwent();
int endpwent();
```

EXPLANATION

Getpwent, getpwuid, and getpwnam each return a pointer to an object with the following structure containing the broken-out fields of a line in the password file.

```
struct  passwd { /* see getpwent(3) */
        char    *pw_name;
        char    *pw_passwd;
        int     pw_uid;
        int     pw_gid;
        int     pw_quota;
        char    *pw_comment;
        char    *pw_gecos;
        char    *pw_dir;
        char    *pw_shell;
};
```

The fields pw_quota and pw_comment are unused; the others have meanings described in **passwd(5)**.

Getpwent reads the next line (opening the file if necessary); setpwent rewinds the file; endpwent closes it.

Getpwuid and getpwnam search from the beginning until a matching uid or name is found (or until EOF is encountered).

FILES

/etc/passwd

SEE ALSO

getlogin(3), getgrent(3), passwd(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

NOTES

All information is contained in a static area so it must be copied if it is to be saved.

gets, fgets - get a string from a stream

SYNTAX

```
#include <stdio.h>

char *gets(s)
char *s;

char *fgets(s, n, stream)
char *s;
FILE *stream;
```

EXPLANATION

Gets reads a string into s from the standard input stream **stdin**. The string is terminated by a newline character, which is replaced in s by a null character. Gets returns its argument.

Fgets reads n-1 characters, or up to a newline character, whichever comes first, from the stream into the string s. The last character read into s is followed by a null character. Fgets returns its first argument.

SEE ALSO

puts(3), getc(3), scanf(3), fread(3), ferror(3)

DIAGNOSTICS

Gets and fgets return the constant pointer **NULL** upon end of file or error.

NOTES

Gets deletes a newline, fgets keeps it, all in the name of backward compatibility.

hypot, cabs - Euclidean distance

SYNTAX

```
#include <math.h>

double hypot(x, y)
double x, y;

double cabs(z)
struct { double x, y;} z;
```

EXPLANATION

Hypot and cabs return

```
sqrt(x*x + y*y),
```

taking precautions against unwarranted overflows.

SEE ALSO

exp(3) for information about sqrt

isalpha, isupper, islower, isdigit, isalnum, isspace, ispunct, isprint, iscntrl, isascii - character classification

SYNTAX
#include <ctype.h>

isalpha(c)

EXPLANATION
These macros classify ASCII-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. Isascii is defined on all integer values; the rest are defined only where isascii is true and on the single non-ASCII value EOF (see **stdio(3)**).

isalpha
c is a letter

isupper
c is an upper case letter

islower
c is a lower case letter

isdigit
c is a digit

isalnum
c is an alphanumeric character

isspace
c is a space, tab, carriage return, newline, or formfeed

ispunct
c is a punctuation character (neither control nor alphanumeric)

isprint
c is a printing character, code 040(8) (space) through 0176 (tilde)

iscntrl
c is a delete character (0177) or ordinary control character (less than 040).

isascii
c is an ASCII character, code less than 0200

j0, j1, jn, y0, y1, yn - bessel functions

SYNTAX

```
#include <math.h>

double j0(x)
double x;

double j1(x)
double x;

double jn(n, x);
double x;

double y0(x)
double x;

double y1(x)
double x;

double yn(n, x)
double x;
```

EXPLANATION

These functions calculate Bessel functions of the first and second kinds for real arguments and integer orders.

DIAGNOSTICS

Negative arguments cause y0, y1, and yn to return a huge negative value and set errno to EDOM.

l3tol, ltol3 - convert between 3-byte integers and long integers

SYNTAX

```
l3tol(lp, cp, n)
long *lp;
char *cp;

ltol3(cp, lp, n)
char *cp;
long *lp;
```

EXPLANATION

L3tol converts a list of n three-byte integers packed into a character string pointed to by cp into a list of long integers pointed to by lp.

Ltol3 performs the reverse conversion from long integers (lp) to three-byte integers (cp).

These functions are useful for file-system maintenance; disk addresses are three bytes long.

SEE ALSO

filsys(5)

itom, madd, msub, mult, mdiv, min, mout, pow, gcd, rpow - multiple precision integer arithmetic

SYNTAX

```
typedef struct { int len; short *val; } mint;

madd(a, b, c)
msub(a, b, c)
mult(a, b, c)
mdiv(a, b, q, r)
min(a)
mout(a)
pow(a, b, m, c)
gcd(a, b, c)
rpow(a, b, c)
msqrt(a, b, r)
mint *a, *b, *c, *m, *q, *r;

sdiv(a, n, q, r)
mint *a, *q;
short *r;

mint *itom(n)
```

EXPLANATION

These routines perform arithmetic on integers of arbitrary length. The integers are stored using the defined type mint. Pointers to a mint should be initialized using the function itom, which sets the initial value to n. After that space is managed automatically by the routines.

madd, msub, mult, assign to their third arguments the sum, difference, and product, respectively, of their first two arguments. mdiv assigns the quotient and remainder, respectively, to its third and fourth arguments. sdiv is like mdiv except that the divisor is an ordinary integer. msqrt produces the square root and remainder of its first argument. rpow calculates a raised to the power b, while pow calculates this reduced modulo m. min and mout do decimal input and output.

The functions are obtained with the loader option -lmp.

DIAGNOSTICS

Illegal operations and running out of memory produce messages and core images.

malloc, free, realloc, calloc - main memory allocator

SYNTAX

```
char *malloc(size)
unsigned size;

free(ptr)
char *ptr;

char *realloc(ptr, size)
char *ptr;
unsigned size;

char *calloc(nelem, elsize)
unsigned nelem, elsize;
```

EXPLANATION

Malloc and free provide a simple general-purpose memory allocation package. Malloc returns a pointer to a block of at least size bytes beginning on a word boundary. The argument to free is a pointer to a block previously allocated by malloc ; this space is made available for further allocation, but its contents are left undisturbed. Needless to say, grave disorder will result if the space assigned by malloc is overrun or if some random number is handed to free.

Malloc allocates the first big enough contiguous reach of free space found in a circular search from the last block allocated or freed, coalescing adjacent free blocks as it searches. It calls sbrk (see **break**(2)) to get more memory from the system when there is no suitable space already free.

Realloc changes the size of the block pointed to by ptr to size bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes. Realloc also works if ptr points to a block freed since the last call of malloc, realloc or calloc. Thus, sequences of free, malloc and realloc can exploit the search strategy of malloc to do storage compaction.

Calloc allocates space for an array of nelem elements of size elsize. The space is initialized to zeros.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

DIAGNOSTICS

Malloc, realloc and calloc return a null pointer (0) if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block. Malloc may be recompiled to check the arena very stringently on every transaction; see the source code.

NOTES

When realloc returns 0, the block pointed to by ptr may be destroyed.

mktemp - make a unique file name

SYNTAX
char *mktemp(template)
char *template;

EXPLANATION
Mktemp replaces template by a unique file name, and returns the address of the template. The template should look like a file name with six trailing X's, which will be replaced with the current process id and a unique letter.

SEE ALSO
getpid(2)

monitor - prepare execution profile

SYNTAX
monitor(lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc)( ), (*highpc)( );
short buffer[ ];

EXPLANATION
An executable program created by 'cc -p' automatically includes calls for monitor with default parameters; monitor needn't be called explicitly except to gain fine control over profiling.

Monitor is an interface to **profil(2)**. Lowpc and highpc are the addresses of two functions; buffer is the address of a (user supplied) array of bufsize short integers. Monitor arranges to record a histogram of periodically sampled values of the program counter, and of counts of calls of certain functions, in the buffer. The lowest address sampled is that of lowpc and the highest is just below highpc. At most nfunc call counts can be kept; only calls of functions compiled with the profiling option -p of cc are recorded. The cc command is provided in the optional 8560 Native Programming Package. For the results to be significant, especially where there are small, heavily used routines, it is suggested that the buffer be no more than a few times smaller than the range of locations sampled.

To profile the entire program, it is sufficient to use

```
extern etext();
...
monitor((int)2, etext, buf, bufsize, nfunc);
```

Etext lies just above all the program text: see **end(3)**.

To stop execution monitoring and write the results on the file mon.out, use

```
monitor(0);
```

then **prof(1)** can be used to examine the results.

FILES
mon.out

SEE ALSO
profil(2), cc
The cc command is provided in the optional 8560 Native Programming Package.

nlist - get entries from name list

SYNTAX

```
#include <a.out.h>
nlist(filename, nl)
char *filename;
struct nlist nl[ ];
```

EXPLANATION

Nlist examines the name list in the given executable output file and selectively extracts a list of values. The name list consists of an array of structures containing names, types and values. The list is terminated with a null name. Each name is looked up in the name list of the file. If the name is found, the type and value of the name are inserted in the next two fields. If the name is not found, both entries are set to 0. See **a.out(5)** for the structure declaration.

This subroutine is useful for examining the system name list kept in the file **/tnix**. In this way programs can obtain system addresses that are up to date.

SEE ALSO

a.out(5)

DIAGNOSTICS

All type entries are set to 0 if the file cannot be found or if it is not a valid namelist.

perror - system error messages

SYNTAX

```
perror(s)
char *s;

int sys_nerr;
char *sys_errlist[];
```

EXPLANATION

Perror produces a short error message on the standard error file describing the last error encountered during a call to the system from a C program. First the argument string s is printed, then a colon, then the message and a new-line. Most usefully, the argument string is the name of the program which incurred the error. The error number is taken from the external variable errno which is set when errors occur but not cleared when non-erroneous calls are made. (**Errno** is described in the Introduction to Section 2 of this manual.)

To simplify variant formatting of messages, the vector of message strings sys errlist is provided; errno can be used as an index in this table to get the message string without the newline. Sys nerr is the number of messages provided for in the table; it should be checked because new error codes may be added to the system before they are added to the table.

SEE ALSO

The Introduction to section 2 of this manual.

pkopen, pkclose, pkread, pkwrite, pkfail - packet driver simulator

SYNTAX

```
char *pkopen(fd)

pkclose(ptr)
char *ptr;

pkread(ptr, buffer, count)
char *ptr, *buffer;

pkwrite(ptr, buffer, count)
char *ptr, *buffer;

pkfail()
```

EXPLANATION

These routines are a user-level implementation of the full-duplex end-to-end communication protocol; the UNIX **pk** packet driver. If fd is a file descriptor open for reading and writing, pkopen carries out the initial synchronization and returns an identifying pointer. The pointer is used as the first parameter to pkread, pkwrite, and pkclose.

Pkread, pkwrite and pkclose behave analogously to read, write and **close**(2). However, a write of zero bytes is meaningful and will produce a corresponding read of zero bytes.

DIAGNOSTICS

Pkfail is called upon persistent breakdown of communication. Pkfail must be supplied by the user.

Pkopen returns a null (0) pointer if packet protocol can not be established.

Pkread returns -1 on end of file, 0 in correspondence with a 0-length write.

NOTES

This simulation of the UNIX packet driver **pk** leaves something to be desired in needing special read and write routines, and in not being inheritable across calls of **exec**(2). Its prime use is on systems that lack pk.
These functions use **alarm**(2); simultaneous use of alarm for other purposes may cause trouble.

plot, openpl, erase, label, line, circle, arc, move, cont, point, linemod, space, closepl - graphics interface

SYNTAX

```
openpl( )

erase( )

label(s) char s[ ];

line(x1, y1, x2, y2)

circle(x, y, r)

arc(x, y, x0, y0, x1, y1)

move(x, y)

cont(x, y)

point(x, y)

linemod(s) char s[ ];

space(x0, y0, x1, y1)

closepl( )
```

EXPLANATION

These subroutines generate graphic output in a relatively device-independent manner. See **plot**(5) for a description of their effect. Openpl must be used before any of the others to open the device for writing. Closepl flushes the output.

String arguments to label and linemod are null-terminated, and do not contain newlines.

Various flavors of these functions exist for different output devices. They are obtained by the following **ld** options:

-lplot
: device-independent graphics stream on standard output for **plot** filters

-l300
: GSI 300 terminal

-l300s
: GSI 300S terminal

-l450
: DASI 450 terminal

-l4014
: Tektronix 4014 terminal

SEE ALSO
plot(5),
graph
The **graph** and **plot** command are provided in the optional 8560 Auxiliary Utilities Package Users Manual. The **ld** command is provided in the optional 8560 Native Programming Package.

popen, pclose - initiate I/O to/from a process

SYNTAX

```
#include <stdio.h>

FILE *popen(command, type)
char *command, *type;

pclose(stream)
FILE *stream;
```

EXPLANATION

The arguments to popen are pointers to null-terminated strings containing respectively a shell command line and an I/O mode, either "r" for reading or "w" for writing. It creates a pipe between the calling process and the command to be executed. The value returned is a stream pointer that can be used (as appropriate) to write to the standard input of the command or read from its standard output.

A stream opened by popen should be closed by pclose, which waits for the associated process to terminate and returns the exit status of the command.

Because open files are shared, a type "r" command may be used as an input filter, and a type "w" as an output filter.

SEE ALSO

pipe(2), fopen(3), fclose(3), system(3), wait(2)

DIAGNOSTICS

Popen returns a null pointer if files or processes cannot be created, or the Shell cannot be accessed.

Pclose returns -1 if stream is not associated with a 'popened' command.

NOTES

Buffered reading before opening an input filter may leave the standard input of that filter mispositioned. Similar problems with an output filter may be forestalled by careful buffer flushing, e.g. with fflush, see **fclose(3)**.

printf, fprintf, sprintf - formated output conversion

SYNTAX

```
#include <stdio.h>

printf(format [, arg ] ... )
char *format;

fprintf(stream, format [, arg ] ... )
FILE *stream;
char *format;

sprintf(s, format [, arg ] ... )
char *s, format;
```

EXPLANATION

Printf places output on the standard output stream stdout. Fprintf places output on the named output stream. Sprintf places 'output' in the string s, followed by the character '\0'.

Each of these functions converts, formats, and prints its arguments after the first under control of the first argument. The first argument is a character string which contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which causes conversion and printing of the next successive arg printf.

Each conversion specification is introduced by the character %. Following the %, there may be

an optional minus sign '-' which specifies left adjustment of the converted value in the indicated field;

an optional digit string specifying a field width; if the converted value has fewer characters than the field width it will be blank-padded on the left (or right, if the left-adjustment indicator has been given) to make up the field width; if the field width begins with a zero, zero-padding will be done instead of blank-padding;

an optional period (".") which serves to separate the field width from the next digit string;

an optional digit string specifying a precision which specifies the number of digits to appear after the decimal point, for e- and f-conversion, or the maximum number of characters to be printed from a string;

the character l specifying that a following **d**, **o**, **x**, or **u** corresponds to a long integer arg. (A capitalized conversion code accomplishes the same thing.)

a character which indicates the type of conversion to be applied.

A field width or precision may be '*' instead of a digit string. In this case an integer arg supplies the field width or precision.

The conversion characters and their meanings are

dox The integer arg is converted to decimal, octal, or hexadecimal notation respectively.

f The float or double arg is converted to decimal notation in the style '[-]ddd.ddd' where the number of d's after the decimal point is equal to the precision specification for the argument. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed.

e The float or double arg is converted in the style '[-]d.ddde±dd' where there is one digit before the decimal point and the number after is equal to the precision specification for the argument; when the precision is missing, 6 digits are produced.

g The float or double arg is printed in style d, in style f, or in style e, whichever gives full precision in minimum space.

c The character arg is printed. Null characters are ignored.

s Arg is taken to be a string (character pointer) and characters from the string are printed until a null character or until the number of characters indicated by the precision specification is reached; however if the precision is 0 or missing all characters up to a null are printed.

u The unsigned integer arg is converted to decimal and printed (the result will be in the range 0 to 65535).

% Print a '%'; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; padding takes place only if the specified field width exceeds the actual width. Characters generated by printf are printed by putc(3).

Examples

To print a date and time in the form 'Sunday, July 3, 10:02', where weekday and month are pointers to null-terminated strings:

```
printf("%s, %s %d, %02d:%02d", weekday, month, day, hour, min);
```

To print to 5 decimals:

```
printf("pi = %.5f", 4*atan(1.0));
```

SEE ALSO

putc(3), scanf(3), ecvt(3)

NOTES

Very wide fields (>128 characters) fail.

putc, putchar, fputc, putw - put character or word on a stream

SYNTAX

```
#include <stdio.h>

int putc(c, stream)
char c;
FILE *stream;

putchar(c)

fputc(c, stream)
FILE *stream;

putw(w, stream)
FILE *stream;
```

EXPLANATION

Putc appends the character c to the named output stream. It returns the character written.

Putchar(c) is defined as putc(c, stdout).

Fputc behaves like putc, but is a genuine function rather than a macro. It may be used to save on object text.

Putw appends word (i.e. **int**) w to the output stream. It returns the word written. Putw neither assumes nor causes special alignment in the file.

The standard stream stdout is normally buffered if and only if the output does not refer to a terminal; this default may be changed by **setbuf(3)**. The standard stream stderr is by default unbuffered unconditionally, but use of freopen (see **fopen(3)**) will cause it to become buffered; setbuf, again, will set the state to whatever is desired. When an output stream is unbuffered information appears on the destination file or terminal as soon as written; when it is buffered many characters are saved up and written as a block. Fflush (see **fclose(3)**) may be used to force the block out early.

SEE ALSO

getc(3), puts(3), printf(3), fread(3)
fopen(3) for **freopen, fclose(3)** for **fflush**

DIAGNOSTICS

These functions return the constant **EOF** upon error. Since this is a good integer, **ferror(3)** should be used to detect putw errors.

NOTES

Because it is implemented as a macro, putc treats a stream argument with side effects improperly. In particular 'putc(c, *f++);' doesn't work sensibly.

puts, fputs - put a string on a stream

SYNTAX

```
#include <stdio.h>

puts(s)
char *s;

fputs(s, stream)
char *s;
FILE *stream;
```

EXPLANATION

Puts copies the null-terminated string s to the standard output stream stdout and appends a newline character.

Fputs copies the null-terminated string s to the named output stream.

Neither routine copies the terminal null character.

SEE ALSO

fopen(3), gets(3), puto(3), printf(3), ferror(3)
fread(3) for fwrite

NOTES

Puts appends a newline, fputs does not, all in the name of backward compatibility.

qsort - quicker sort

SYNTAX

```
qsort(base, nel, width, compar)
char *base;
int (*compar)( );
```

EXPLANATION

Qsort is an implementation of the quicker-sort algorithm. The first argument is a pointer to the base of the data; the second is the number of elements; the third is the width of an element in bytes; the last is the name of the comparison routine to be called with two arguments which are pointers to the elements being compared. The routine must return an integer less than, equal to, or greater than 0 according as the first argument is to be considered less than, equal to, or greater than the second.

rand, srand - random number generator

SYNTAX

```
srand(seed)
int seed;

rand( )
```

EXPLANATION

Rand uses a multiplicative congruential random number generator with period 2**32 to return successive pseudo-random numbers in the range from 0 to (2**15)-1.

The generator is reinitialized by calling srand with 1 as argument. It can be set to a random starting point by calling srand with whatever you like as argument.

scanf, fscanf, sscanf - formated input conversion

SYNTAX

```
#include <stdio.h>

scanf(format [ , pointer ] . . .  )
char *format;

fscanf(stream, format [ , pointer ] . . .  )
FILE *stream;
char *format;

sscanf(s, format [ , pointer ] . . .  )
char *s, *format;
```

EXPLANATION

Scanf reads from the standard input stream stdin. Fscanf reads from the named input stream. Sscanf reads from the character string s. Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects as arguments a control string format, described below, and a set of pointer arguments indicating where the converted input should be stored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. Blanks, tabs or newlines, which match optional white space in the input.

2. An ordinary character (not %) which must match the next character of the input stream.

3. Conversion specifications, consisting of the character %, an optional assignment suppressing character *, an optional numerical maximum field width, and a conversion character.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by *. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. The following conversion characters are legal:

% a single '%' is expected in the input at this point; no assignment is done.

d a decimal integer is expected; the corresponding argument should be an integer pointer.

o an octal integer is expected; the corresponding argument should be a integer pointer.

x a hexadecimal integer is expected; the corresponding argument should be an integer pointer.

s a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating '\0', which will be added. The input field is terminated by a space character or a newline.

c a character is expected; the corresponding argument should be a character pointer. The normal skip over space characters is suppressed in this case; to read the next non-space character, try '%1s'. If a field width is given, the corresponding argument should refer to a character array, and the indicated number of characters is read.

e
f a floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a float. The input format for floating point numbers is an optionally signed string of digits possibly containing a decimal point, followed by an optional exponent field consisting of an E or e followed by an optionally signed integer.

[indicates a string not to be delimited by space characters. The left bracket is followed by a set of characters and a right bracket; the characters between the brackets define a set of characters making up the string. If the first character is not circumflex (^), the input field is all characters until the first character not in the set between the brackets; if the first character after the left bracket is ^, the input field is all characters until the first character which is in the remaining set of characters between the brackets. The corresponding argument must point to a character array.

The conversion characters **d**, **o** and **x** may be capitalized or preceded by **l** to indicate that a pointer to **long** rather than to **int** is in the argument list. Similarly, the conversion characters **e** or **f** may be capitalized or preceded by **l** to indicate a pointer to **double** rather than to **float**. The conversion characters **d**, **o** and **x** may be preceded by **h** to indicate a pointer to **short** rather than to **int**.

The scanf functions return the number of successfully matched and assigned input items. This can be used to decide how many input items were found. The constant **EOF** is returned upon end of input; note that this is different from 0, which means that no conversion was done; if conversion was intended, it was frustrated by an inappropriate character in the input.

For example, the call

```
int i; float x; char name[50];
scanf("%d%f%s", &i, &x, name);
```

with the input line

```
25   54.32E-1  thompson
```

will assign to i the value 25, x the value 5.432, and name will contain 'thompson\0'. Or,

```
int i; float x; char name[50];
scanf("%2d%f%*d%[1234567890]", &i, &x, name);
```

with input

```
56789 0123 56a72
```

will assign 56 to i, 789.0 to x, skip '0123', and place the string '56\0' in name. The next call to getchar will return 'a'.

SEE ALSO
atof(3), getc(3), printf(3)

DIAGNOSTICS
The scanf functions return **EOF** on end of input, and a short count for missing or illegal data items.

NOTES
The success of literal matches and suppressed assignments is not directly determinable.

setbuf - assign buffering to a stream

SYNTAX

```
#include <stdio.h>

setbuf(stream, buf)
FILE *stream;
char *buf;
```

EXPLANATION

Setbuf is used after a stream has been opened but before it is read or written. It causes the character array buf to be used instead of an automatically allocated buffer. If buf is the constant pointer NULL, input/output will be completely unbuffered.

A manifest constant BUFSIZ tells how big an array is needed:

```
char buf[BUFSIZ];
```

A buffer is normally obtained from **malloc(3)** upon the first getc or **putc(3)** on the file, except that output streams directed to terminals, and the standard error stream stderr are normally not buffered.

SEE ALSO

fopen(3), getc(3), putc(3), malloc(3)

setjmp, longjmp - non-local goto

SYNTAX

```
#include <setjmp.h>

setjmp(env)
jmp_buf env;

longjmp(env, val)
jmp_buf env;
```

EXPLANATION

These routines are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

Setjmp saves its stack environment in env for later use by longjmp. It returns value 0.

Longjmp restores the environment saved by the last call of setjmp. It then returns in such a way that execution continues as if the call of setjmp had just returned the value val to the function that invoked setjmp, which must not itself have returned in the interim. All accessible data have values as of the time longjmp was called.

SEE ALSO

signal(2)

sin, cos, tan, asin, acos, atan, atan2 - trigonometric functions

SYNTAX

```
#include <math.h>

double sin(x)
double x;

double cos(x)
double x;

double asin(x)
double x;

double acos(x)
double x;

double atan(x)
double x;

double atan2(x, y)
double x, y;
```

EXPLANATION

Sin, cos and tan return trigonometric functions of radian arguments. The magnitude of the argument should be checked by the caller to make sure the result is meaningful.

Asin returns the arc sin in the range -pi/2 to pi/2.

Acos returns the arc cosine in the range 0 to pi.

Atan returns the arc tangent of x in the range -pi/2 to pi/2.

Atan2 returns the arc tangent of x/y in the range -pi to pi.

DIAGNOSTICS

Arguments of magnitude greater than 1 cause asin and acos to return value 0; errno is set to EDOM. The value of tan at its singular points is a huge number, and errno is set to ERANGE.

NOTES

The value of tan for arguments greater than about 2**31 is garbage.

sinh, cosh, tanh - hyperbolic functions

SYNTAX

```
#include <math.h>

double sinh(x)
double x;

double cosh(x)
double x;

double tanh(x)
double x;
```

EXPLANATION

These functions compute the designated hyperbolic functions for real arguments.

DIAGNOSTICS

Sinh and cosh return a huge value of appropriate sign when the correct value would overflow.

sleep - suspend execution for interval

SYNTAX
sleep(seconds)
unsigned seconds;

EXPLANATION
The current process is suspended from execution for the number of seconds specified by the argument. The actual suspension time may be up to 1 second less than that requested, because scheduled wakeups occur at fixed 1-second intervals, and an arbitrary amount longer because of other activity in the system.

The routine is implemented by setting an alarm clock signal and pausing until it occurs. The previous state of this signal is saved and restored. If the sleep time exceeds the time to the alarm signal, the process sleeps only until the signal would have occurred, and the signal is sent 1 second later.

SEE ALSO
alarm(2), pause(2)

stdio - standard buffered input/output package

SYNTAX
#include <stdio.h>

FILE *stdin;
FILE *stdout;
FILE *stderr;

EXPLANATION
The functions described in Sections 3S constitute an efficient user-level buffering scheme. The in-line macros getc and **putc(3)** handle characters quickly. The higher level routines gets, fgets, scanf, fscanf, fread, puts, fputs, printf, fprintf, fwrite all use getc and putc; they can be freely intermixed.

A file with associated buffering is called a stream, and is declared to be a pointer to a defined type **FILE**. **Fopen(3)** creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. There are three normally open streams with constant pointers declared in the include file and associated with the standard open files:

stdin
    standard input file
stdout
    standard output file
stderr
    standard error file

A constant 'pointer' **NULL** (0) designates no stream at all.

An integer constant **EOF** (-1) is returned upon end of file or error by integer functions that deal with streams.

Any routine that uses the standard input/output package must include the header file <stdio.h> of pertinent macro definitions. The functions and constants mentioned in sections labeled 3S are declared in the include file and need no further declaration. The constants, and the following 'functions' are implemented as macros; redeclaration of these names is perilous: getc, getchar, putc, putchar, feof, ferror, fileno.

SEE ALSO
open(2), close(2), read(2), write(2)

DIAGNOSTICS
The value **EOF** is returned uniformly to indicate that a **FILE** pointer has not been initialized with fopen, input (output) has been attempted on an output (input) stream, or a **FILE** pointer designates corrupt or otherwise unintelligible **FILE** data.

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, index, rindex - string operations

SYNTAX

```
char *strcat(s1, s2)
char *s1, *s2;

char *strncat(s1, s2, n)
char *s1, *s2;

strcmp(s1, s2)
char *s1, *s2;

strncmp(s1, s2, n)
char *s1, *s2;

char *strcpy(s1, s2)
char *s1, *s2;

char *strncpy(s1, s2, n)
char *s1, *s2;

strlen(s)
char *s;

char *index(s, c)
char *s, c;

char *rindex(s, c)
char *s;
```

EXPLANATION

These functions operate on null-terminated strings. They do not check for overflow of any receiving string.

Strcat appends a copy of string s2 to the end of string s1. Strncat copies at most n characters. Both return a pointer to the null-terminated result.
Strcmp compares its arguments and returns an integer greater than, equal to, or less than 0, according as s1 is lexicographically greater than, equal to, or less than s2. Strncmp makes the same comparison but looks at at most n characters.
Strcpy copies string s2 to s1, stopping after the null character has been moved. Strncpy copies exactly n characters, truncating or null-padding s2; the target may not be null-terminated if the length of s2 is n or more. Both return s1.
Strlen returns the number of non-null characters in s.
Index (rindex) returns a pointer to the first (last) occurrence of character c in string s, or zero if c does not occur in the string.

NOTES

Strcmp uses native character comparison, which is signed on PDP11's, unsigned on other machines.

swab - swap bytes

SYNTAX

```
swab(from, to, nbytes)
char *from, *to;
```

EXPLANATION

Swab copies nbytes bytes pointed to by from to the position pointed to by to, exchanging adjacent even and odd bytes. It is useful for carrying binary data between PDP11's and other machines. Nbytes should be even.

system - issue a shell command

SYNTAX
system(string)
char *string;

EXPLANATION
System causes the string to be given to **sh(1)** as input as if the string had been typed as a command at a terminal. The current process waits until the shell has completed, then returns the exit status of the shell.

SEE ALSO
popen(3), exec(2), wait(2)

DIAGNOSTICS
Exit status 127 indicates the shell couldn't be executed.

ttyname, isatty, ttyslot - find name of a terminal

SYNTAX
char *ttyname(fildes)

isatty(fildes)

ttyslot()

EXPLANATION
Ttyname returns a pointer to the null-terminated path name of the terminal device associated with file descriptor fildes.

Isatty returns 1 if fildes is associated with a terminal device, 0 otherwise.

Ttyslot returns the number of the entry in the **ttys(5)** file for the control terminal of the current process.

FILES
/dev/*
/etc/ttys

SEE ALSO
ioctl(2), ttys(5)

DIAGNOSTICS
Ttyname returns a null pointer (0) if fildes does not describe a terminal device in directory '/dev'.

Ttyslot returns 0 if '/etc/ttys' is inaccessible or if it cannot determine the control terminal.

NOTES
The return value points to static data whose content is overwritten by each call.

ungetc - push character back into input stream

SYNTAX
#include <stdio.h>

ungetc(c, stream)
FILE *stream;

EXPLANATION
Ungetc pushes the character c back on an input stream. That character will be returned by the next getc call on that stream. Ungetc returns c.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered. Attempts to push EOF are rejected.

Fseek(3) erases all memory of pushed back characters.

SEE ALSO
getc(3), setbuf(3), fseek(3)

DIAGNOSTICS
Ungetc returns **EOF** if it can't push a character back.

Section 4

SPECIAL FILES

INTRODUCTION

This section discusses data transfer and access to/from special files. **Special files** are hardware input/output devices that appear to the user as regular files, making inter-device file/data transfer within TNIX straightforward. Terminals, printers, and disk drives are examples of some special files.

NOTATION

You will see an integer in parentheses after some of the command names. This indicates that the command is described in the specified section of this manual. For example, **perror(3)** indicates that the **perror** standard subroutine is described in Section 3 of this manual.

Here is a list of the special files described in this section:

aux	-- line printer	-- 4-2
fd	-- flexible disk drive	-- 4-3
gpib	-- interface to GPIB-compatible devices	-- 4-4
hd	-- winchester fixed disks	-- 4-5
hsi, hsix	-- high speed interface protocols	-- 4-7
kmem	-- memory	-- 4-8
mem	-- memory	-- 4-8
null	-- data sink	-- 4-9
tty	-- general terminal interface	-- 4-10

aux - line printer

EXPLANATION
Aux1 and Aux2 are special files which access the line printers. They will provide tab to space substitution for printers without hardware tabs, and will provide end-of-line substitution. The most common use of end-of-line substitution is to change a <CR><LF> sequence to just <CR> or <LF>.

The command **slp(1)** can be used to set the end-of-line and tab options.

The ioctls which are needed for setting the end-of-line string and turning the tab expansion on and off are LPSETNL, LPTABON, and LPTABOFF, respectively. The formats of these requests are:

```
#include <lp.h>
ioctl(fildes,LPSETNL,buf)
char buf[8];

and

#include <lp.h>
ioctl(fildes,LPTABON)

and

#include <lp.h>
ioctl(fildes,LPTABOFF)
```

where buf is a null-terminated string to substitute for the end-of-line character, linefeed. The string may be up to 8 characters long. Null characters may be inserted in the string by setting the high bit. This high bit will not be sent to the line printer.

FILES
/dev/aux1
/dev/aux2

SEE ALSO
slp(1)

fd - flexible disk

EXPLANATION
The file **fd0** refers to the flexible disk drive. The **fd0** file accesses the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records.

There is also a 'raw' interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation; therefore, raw I/O is considerably more efficient (for multiple block buffers when many words are transmitted). The name of the raw **fd0** file is **rfd0**. In raw I/O, the buffer must begin on a word boundary.

The **ioctl(2)** device control system calls are available for the raw device. GETSIZ and PMSCMD are functions executed by the **ioctl** system call. The GETSIZ **ioctl** request returns the maximum block number on the device into a long integer. The PMSCMD **ioctl** request sends a direct command to the low-level Winchester disk controller and returns the response in the same location.

The GETSIZ and PMSCMD **ioctl** requests use the form:

```
#include <disk.h>
ioctl(fildes,GETSIZ,buf)
long *buf;
```

and

```
#include <disk.h>
ioctl(fildes,PMSCMD,buf)
unsigned buf[10];
```

If the drive number in the PMSCMD request does not correspond to one of the disk drives associated with the open disk device, **ioctl** returns an error. This keeps people from accessing any location on the fixed disk by opening the flexible disk or another fixed disk as a device and issuing an **ioctl** call.

FILES
/dev/fd0
/dev/rfd0

SEE ALSO
hd(4), ioctl(2)

NOTES
In raw I/O, **read(2)** and **write(2)** truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write, and **lseek(2)** should always deal in 512-byte multiples.

gpib - Interface to GPIB-compatible devices

EXPLANATION

This discussion describes four different types of special files that are compatible with IEEE Standard 488-1978 (commonly known as the General Purpose Interface Bus, or GPIB). This discussion assumes that you are familiar with the contents and nomenclature of the standard. You may obtain a copy of the standard from the Institute of Electrical and Electronic Engineers, Inc.; 345 East 47th St. New York, NY 10017.

Each GPIB special file type has characteristics that tailor it to specific GPIB interfacing applications. The file types are designated by the associated device driver routine name (known as a GD):

magtape — magtape devices provide access to Dylon Corporation's 2001/9001 Series and 4000A series magnetic tape drives.

instr — instr devices provide access to most GPIB-compatible instruments. These devices have facilities that reduce GPIB programming burdens, such as automatic address generation and serial poll control.

direct — direct devices provide direct access to the GPIB itself. This type of device can be used to control system-wide bus resources, such as the IFC(L) and REN(L) lines, or GPIB-compatible instruments that can't function with the instr interface.

monitor — monitor is the GPIB Interface itself. Only one monitor device (/dev/gpib) can exist in the system. ldgpib(1) establishes the monitor if /dev/gpib does not already exist.

You may have up to 14 GPIB devices (excluding /dev/gpib) defined on the system at a time, using any mixture of magtape, instr, and direct devices. However, you will normally want only one of the devices to be a direct device.

Device Access

All GPIB devices are accessed by C programs, either through system calls or subroutines. TNIX contains several utility programs that access magtape devices, such as **dump**(1) and **settape**(1). You must create your own programs (with the optional 8560 Series Native Programming Package) to access instr and direct devices.

Five system calls provide access to GPIB devices: **open**(2), **close**(2), **read**(2), **write**(2), and **ioctl**(2).

hd - Winchester fixed disks

EXPLANATION
The files [r]**hd*** (for example, hd0, rhd0, and hd03) refer to the Winchester fixed disk drives.

Files without a leading **r** (e.g., **hd03**) access the disks via the system's normal buffering mechanism and may be read and written without regard to physical disk records. Files containing a leading **r** (e.g., **rhd03**) are the 'raw' devices, providing direct transmission between a disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation; therefore, raw I/O is considerably more efficient when many words are transmitted.

Because a TNIX disk device can include one or more physical disk drives, the two final digits of the filename are the drive numbers of the first and last fixed disk drives associated with the TNIX device. The second digit is omitted if the TNIX device occupies only one fixed disk drive. For example, the special file /dev/hd1 occupies one fixed disk drive (drive 1); the special file /dev/hd02 includes the first three fixed disk drives (0 -- 2, inclusive).

The minor device numbers of an **hd** device reflect this organization: the low two bits (0 and 1) give the first drive of the TNIX disk device; the next bit(2) is always zero; the next two bits (3 and 4) give the final drive of the TNIX disk device.

The **ioctl** system calls are available for the raw disk devices. GETSIZ and PMSCMD are functions executed by the **ioctl** system call. GETSIZ will return the maximum block number on the disk device. PMSCMD will give a direct command to the Winchester disk controller and will return the response in the same location.

The GETSIZ and PMSCMD requests use the following form:

```
#include <disk.h>
ioctl(fildes,GETSIZ,buf)
long *buf;
```

and

```
#include <disk.h>
ioctl(fildes,PMSCMD,buf)
unsigned buf[10];
```

If the drive number in the PMSCMD request does not correspond to one of the physical disk drives associated with the open disk device, the **ioctl** system call returns an error. This keeps people from accessing any location on the fixed disk by opening the flexible disk or another fixed disk as a device and issuing an **ioctl** system call.

FILES
/dev/[r]hd*

SEE ALSO
fd(4), ioctl(2)

NOTES
In raw I/O, **read**(2) and **write**(2) truncate file offsets to 512-byte block boundaries, and **write** scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write, and **lseek**(2) should always deal in 512-byte multiples.

In raw I/O, the buffer must begin on a word boundary.

The result of a "get max disk block number" PMSCMD request will differ from the result of a GETSIZ operation. This is because:

- the last track of every fixed disk is reserved for diagnostics and therefore is not included in what GETSIZ returns,
- the Winchester disk controller thinks that each logical device occupies only one physical drive.

hsi, hsix - high speed interface protocols

EXPLANATION
Hsi is the device name for the HSI protocol. The details of the HSI protocol are not available at this printing. The HSI protocol allows an error free transfer of blocks of data of up to 600 bytes in a block. This protocol is used in communication with the 8540. The device hsix uses the HSI protocol with a few changes to allow an end-of-file indication (a 0 length read) to be sent across the link. When the hsix device is closed after being open for write only, a zero-length chunk is sent to indicate the end of file.

The hsix device may be used for transferring data from one 8560 to another. For example, the command 'cat </dev/hsix3 >filename' on one 8560, with the command 'cat filename >/dev/hsix3' on another 8560, would transfer a file from one 8560 to the other through the HSI cable connected to port 3 on both 8560s. The zero length block sent when the sending 8560 closes the hsix device will correctly terminate the command on the receiving 8560. The block size sent over the hsix device can be up to 598 bytes. The ports being used for this HSI transfer must be either strapped for HSI, or forced to HSI mode with a **stty(1)** command, such as 'stty IU >/dev/ttyn'.

FILES
/dev/hsi, /dev/hsix

SEE ALSO
stty(1), tty(4)

mem, kmem - memory

EXPLANATION
Mem is a special file that is an image of the 8560's memory. It may be used, for example, to examine, and even to patch the system. **Kmem** is the same as **mem** except that kernel virtual memory rather than physical memory is accessed.

Byte addresses are interpreted as memory addresses. References to non-existent locations return errors.

Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

On the 8560, the I/O page begins at location 0160000 of **kmem** and per-process data for the current process begins at 0140000.

FILES
/dev/mem, /dev/kmem

NOTES
On the 8560, memory files are accessed one byte at a time, an inappropriate method for some device registers.

null - data sink

EXPLANATION
Data written on a null special file is discarded. This is useful when you don't care to see the output of commands used in a shell program (for example, **testing** the return status of **fgrep**) and you don't care what the result of the **fgrep** is.

Reads from a null special file always return 0 bytes. This provides an easy way to clear files at the start of a shell program (for example, cat /dev/null >logfile).

FILES
/dev/null

tty - general terminal interface

EXPLANATION

This discussion describes both a particular special file, and the general nature of the terminal interface.

The file /dev/tty is, in each process, a synonym for the control terminal associated with that process. It is useful for programs that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand a file name for output, when typed output is desired and it is tiresome to find out which terminal is currently in use.

As for terminals in general: all of the low-speed asynchronous communications ports use the same general interface, no matter what hardware is involved. The remainder of this section discusses the common features of the interface.

When a terminal file is opened, it causes the process to wait until a connection is established. In practice, user programs seldom open these files; they are opened by **init(1)** and become a user's input and output file. The very first terminal file open in a process becomes the control terminal for that process. The control terminal plays a special role in handling quit or interrupt signals, as discussed below. The control terminal is inherited by a child process during a fork, even if the control terminal is closed. The set of processes that thus share a control terminal is called a process group. All members of a process group receive certain signals together (see ETX below and **kill(2)**).

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the system's character input buffers become completely choked, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program.

Normally, terminal input is processed in units of lines. This means that a program attempting to read will be suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not however necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information. There are special modes, discussed below, that permit the program to read each character as typed without waiting for a full line.

During input, erase and kill processing is done normally. By default, the character <bs> erases the last character typed, except that it will not erase beyond the beginning of a line or an EOT. By default, the character <^U> kills the entire line up to the point where it was typed, but not beyond an EOT. Both these characters operate on a keystroke basis independently of any backspacing or tabbing that may have been done. Either <^U> or <bs> may be entered literally by preceding it by '\'; the erase or kill character remains, but the '\' disappears.

The character <^R> will retype the line as input so far.

The character <^K> will retype successive characters of the previous line.

Certain ASCII control characters have special meaning. These characters are not passed to a reading program except in raw mode, where they lose their special meaning. Also, it is possible to change these characters from the default, as described below.

EOT (Control-D) may be used to generate an end of file from a terminal. When an EOT is received, all the characters waiting to be read are immediately passed to the program, without waiting for a newline, and the EOT is discarded. Thus if there are no characters waiting, which is to say the EOT occurred at the beginning of a line, zero characters will be passed back, and this is the standard end-of-file indication.

ETX (Control-C) is not passed to a program but generates an interrupt signal which is sent to all processes with the associated control terminal. Normally each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location. See **signal(2)**.

FS (Control-\ or control-shift-L) generates the quit signal. Its treatment is identical to the interrupt signal except that unless a receiving process has made other arrangements it will not only be terminated but a memory image file will be generated.

DC3 (Control-S) delays all printing on the terminal until a DC1 is typed in (or when an interrupt or quit signal is generated by typing ETX or FS).

DC1 (Control-Q) restarts printing after DC3 without generating any input to a program.

When the carrier signal from the dataset drops (usually because the user has hung up his terminal) a hangup signal is sent to all processes with the terminal as control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hangup signal is ignored, any read returns with an end-of-file indication. Thus programs that read a terminal and test for end-of-file on their input can terminate appropriately when hung up on.

When one or more characters are written, they are actually transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive. When a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold the program is resumed. The EOT character is not transmitted (except in raw mode) to prevent terminals that respond to it from hanging up.

Several ioctl(2) calls apply to terminals. Most of them use the following structure, defined in <tiop.h>:

```
struct sgttyb {
        char sg_ispeed;
        char sg_ospeed;
        char sg_erase;
        char sg_kill;
        int  sg_flags;
};
```

This is the structure which is pointed to by argp. The sg_ispeed field describes the input and output speeds of the device according to the following table. The sg_ospeed field is no longer used. Symbolic values in this table are as defined in <tiop.h>.

B300	7	300 baud
B600	8	600 baud
B1200	9	1200 baud
B2400	11	2400 baud
B4800	12	4800 baud
B9600	13	9600 baud

The sg_erase and sg_kill fields of the argument structure specify the erase and kill characters, respectively. Setting the high bit of the erase or kill character cause the erase or kill function to be the defaults: ^H and ^U.

The sg_flags field contains several bits that determine the system's treatment of the terminal. Bit fields within sg_flags are followed by the values it can take on.

```
TANDEM  03
        possible values:
        00 off
        01 XON/XOFF flagging
        02 DTR flagging
CBREAK  04
ECHO    010
CRMOD   020
RAW     040
XTABS   0100
ARESM   0200
CTSFLG  0400
IPRTY   07000
        possible values:
        00000 don't care
        01000 even parity
        02000 odd
        03000 no parity (zero)
        04000 mark (one)
        05000 data (8th bit not stripped)
OPRTY   070000
        possible values:
        000000 don't care
        010000 even parity
        020000 odd
        030000 no parity (zero)
        040000 mark (one)
        050000 data (8th bit not stripped)
CHNGED  0100000
```

Characters with the wrong parity are ignored.

In raw mode, every character is passed immediately to the program without waiting until a full line has been typed. There is no special meaning associated with any character. For example, no erase or kill processing is done; the end-of-file indicator (EOT), the interrupt character (DEL) and the quit character (FS) are not treated specially. There is no echoing, and no replacement of one character for another; characters are a full 8 bits for both input and output (parity is up to the program).

CRMOD causes input carriage returns to be turned into newlines; input of either CR or LF causes a settable string of up to eight characters to be output in place of the newline character. A typical string would be "<CR><LF>".

CBREAK is a sort of half-cooked mode. Programs can read each character as soon as typed, instead of waiting for a full line, but quit, DC1, DC3, and interrupt work; CRMOD, XTABS, ECHO, and parity work normally. On the other hand, there is no erase or kill, and no special treatment of the escape character, <^R>, <^K>, or EOT.

ARESM turns on auto resume, which causes any input character to resume the output if it has been suspended by a DC3.

XTABS turns on tab expansion, which causes tab characters to be expanded to the appropriate number of spaces on output. Tabs are fixed at every eight columns, starting at column 9.

The TANDEM field activates flow-control on input. When enabled, the IOP signals whatever is on the terminal line to stop sending when its buffers begin to get full, and signals it to start again when its buffer gets sufficiently empty.

CTSFLG turns on CTS flagging, which causes output to be stopped when CTS goes low and to be restarted when CTS goes high.

CHNGED is set when the large characteristic table has been changed and cleared when the data structure described above is set (since this resets the characteristic table to a standard state).

Several ioctl calls have the form:

```
#include <tiop.h>
ioctl(fildes,code,arg)
struct sgttyb *arg;
```

The applicable codes are:

TIOPGETP - Fetch the parameters associated with the terminal, and store in the pointed-to sgttyb structure described earlier.

TIOPSETP - Set the parameters according to the pointed-to sgttyb structure. The interface delays until output is quiescent, then throws away any unread characters, before changing the modes.

TIOPSETN - Set the parameters but do not delay or flush input. Switching out of RAW or CBREAK mode may cause some garbage input.

TIOCGETP, TIOCSETP, and TIOCSETN perform the same functions as TIOPGETP, TIOPSETP, and TIOPSETN, but interpret the flag word of the sgttyb structure as the old UNIX tty driver did. These are included for compatibility with old object code.

With the following four codes the arg is ignored.

TIOIU - Turn on support for block mode communication (HSI) with an IU . This mode may not be turned off once it is enabled.

TIOCEXCL - Set "exclusive-use" mode: no further opens are permitted until the file has been closed.

TIOCNXCL - Turn off "exclusive-use" mode.

TIOCHPCL - When the file is closed for the last time, hang up the terminal.

TIOCNHPCL - When the file is closed for the last time, do not hang up the terminal.

TIOCFLUSH - All characters waiting in input or output queues are flushed.

The large terminal characteristic table may be set with the following calls:

TIOTSET - Fill in the table with the values at arg. Note that this also sets the CHNGED bit in the sg_flags field of the sgttyb structure, to alert the user when s/he does a TIOPGETP that the table has been changed. This bit is cleared by a TIOPSET request. The TIOTSET command can change the function of any input character. Thus, all of the characters with special meaning like DC1, DC3, <^R>, <^K>, EOT, and <^C>, may be changed to another character or removed.

TIOTGET - Fetch the values in the table and store them at arg.

INCLUDE FILES
<tiop.h>

<iopcmd.h>

<trm_def.h>

Section 5

FILE FORMATS

INTRODUCTION

This section describes some of the data structures, file structures, and file formats used with the TNIX operating system and with TNIX programming tools: Information necessary for system accounting, such as the password layout, is provided in this section. Disk addressing conventions used by the operating system are also described in this section.

NOTATION

You will see an integer in parentheses after some of the command names. This indicates that the command is described in the specified section of this manual. For example, **perror(3)** indicates that the **perror** standard subroutine is described in Section 3 of this manual.

Here is a list of all the file formats described in this section:

a.out	-- assembler and link editor output	-- 5-2
ar	-- archive (library) file format	-- 5-6
banproto	-- spooler banner page prototype files	-- 5-7
checklist	-- list of normally mounted file systems	-- 5-9
core	-- format of core image file	-- 5-10
cvt	-- structure of kernel CVT table	-- 5-11
ddate	-- incremental dump format	-- 5-15
dir	-- format of directories	-- 5-13
dist	-- format of a distribution floppy	-- 5-14
dump	-- incremental dump format	-- 5-15
environ	-- user environment	-- 5-18
fbr	-- file backup and restore format	-- 5-19
filsys	-- format of file system volume	-- 5-21
flblk	-- format of file system volume	-- 5-21
gpib	-- GPIB interface data files	-- 5-24
group	-- group file	-- 5-25
ino	-- format of file system volume	-- 5-21
mtab	-- mounted file system table	-- 5-26
passwd	-- password file	-- 5-27
plot	-- graphics interface	-- 5-28
termcap	-- terminal capability data base	-- 5-30
ttys	-- terminal initialization data	-- 5-39
utmp,wtmp	-- login records	-- 5-40

a.out - assembler and link editor output

SYNTAX

```
#include <a.out.h>
```

EXPLANATION

A.out is the output file of the assembler as and the link editor ld. The as and the ld commands are included in the optional 8560 Series Native Programming Package. Both programs make a.out executable if there were no errors and no unresolved external references. Layout information as given in the include file for the LSI 11/23 is:

```
/*
 * Header prepended to each a.out file.
 */
struct exec {
        long    a_magic;        /* magic number */
unsigned long   a_text;         /* size of text segment */
unsigned long   a_data;         /* size of initialized data */
unsigned long   a_bss;          /* size of uninitialized data */
unsigned long   a_syms;         /* size of symbol table */
unsigned long   a_entry;        /* entry point */
unsigned long   a_trsize;       /* size of text relocation */
unsigned long   a_drsize;       /* size of data relocation */
};

#define OMAGIC  0407            /* old impure format */
#define StAGIC  0410            /* read-only text */
#define StAGIC  0410            /* read-only text */
#define ZMAGIC  0413            /* demand load format */

/*
 * Macros which take exec structures as arguments and tell whether
 * the file has a reasonable magic number or offsets to text|symbols|strings.
 */
#define N_BADMAG(x) \
    (((x).a_magic)!=OMAGIC && ((x).a_magic)!=StAGIC && ((x).a_magic)!=ZMAGIC)

#define N_TXTOFF(x)     ((x).a_magic==ZMAGIC ? 1024 : sizeof (struct exec))
#define N_SYMOFF(x) \
        (N_TXTOFF(x) + (x).a_text+(x).a_data + (x).a_trsize+(x).a_drsize)
#define N_STROFF(x)     (N_SYMOFF(x) + (x).a_syms)

/*
 * Format of a relocation datum.
 */
struct relocation_info {
        int     r_address;      /* address which is relocated */
unsigned int    r_symbolnum:24, /* local symbol ordinal */
                r_pcrel:1,      /* was relocated pc relative already */
                r_length:2,     /* 0=byte, 1=word, 2=long */
                r_extern:1,     /* does not include value of sym referenced */
                :4;             /* nothing, yet */
};
```

```
/*
 * Format of a symbol table entry; this file is included by <a.out.h>
 * and should be used if you aren't interested the a.out header
 * or relocation information.
 */
struct  nlist {
        union {
                char    *n_name;        /* for use when in-core */
                long    n_strx;         /* index into file string table */
        } n_un;
unsigned char   n_type;         /* type flag, i.e. N_TEXT etc; see below */
        char    n_other;        /* unused */
        short   n_desc;         /* see <stab.h> */
unsigned long   n_value;        /* value of this symbol (or sdb offset) */
};
#define n_hash  n_desc          /* used internally by ld */

/*
 * Simple values for n_type.
 */
#define N_UNDF  0x0             /* undefined */
#define N_ABS   0x2             /* absolute */
#define N_TEXT  0x4             /* text */
#define N_DATA  0x6             /* data */
#define N_BSS   0x8             /* bss */
#define N_COMM  0x12            /* common (internal to ld) */
#define N_FN    0x1f            /* file name symbol */

#define N_EXT   01              /* external bit, or'ed in */
#define N_TYPE  0x1e            /* mask for all the type bits */

/*
 * Sdb entries have some of the N_STAB bits set.
 * These are given in <stab.h>
 */
#define N_STAB  0xe0            /* if any of these bits set, a SDB entry */

/*
 * Format for namelist values.
 */
#define N_FORMAT        "%08x"
```

The file has four sections: a header, the program and data text, relocation information, and a symbol table (in that order). The last two may be empty if the program was loaded with the '-s' option of **ld** or if the symbols and relocation have been removed by **strip**. The **ld** and the **strip** commands are included in the optional 8560 Series Native Programming Package.

In the header the sizes of each section are given in bytes, but are even. The size of the header is not included in any of the other sizes.

When an a.out file is loaded into core for execution, three logical segments are set up: the text segment, the data segment (with uninitialized data, which starts off as all 0, following initialized), and a stack. The text

segment begins at 0 in the core image; the header is not loaded. If the magic number in the header is 0407(octal), it indicates that the text segment is not to be write-protected and shared, so the data segment is immediately contiguous with the text segment. If the magic number is 0410, the data segment begins at the first 0 mod 8K byte boundary following the text segment, and the text segment is not writable by the program; if other processes are executing the same file, they will share the text segment. If the magic number is 0411, the text segment is again pure, write-protected, and shared, and moreover instruction and data space are separated; the text and data segment both begin at location 0. If the magic number is 0405, the text segment is overlaid on an existing (0411 or 0405) text segment and the existing data segment is preserved.

The stack will occupy the highest possible locations in the core image: from 0177776(octal) and growing downwards. The stack is automatically extended as required. The data segment is only extended as requested by **brk(2)**.

The start of the text segment in the file is 020(octal). The start of the data segment is $020+S_t$ (the size of the text) the start of the relocation information is $020+S_t+S_d$. The start of the symbol table is $020+2(S_t+S_d)$ if the relocation information is present, $020+S_t+S_d$ if not.

The layout of a symbol table entry and the principal flag values that distinguish symbol types are given in the include file. Other flag values may occur if an assembly language program defines machine instructions.

If a symbol's type is undefined external, and the value field is non-zero, the symbol is interpreted by the loader ld as the name of a common region whose size is indicated by the value of the symbol.

The value of a word in the text or data portions which is not a reference to an undefined external symbol is exactly that value which will appear in core when the file is executed. If a word in the text or data portion involves a reference to an undefined external symbol, as indicated by the relocation information for that word, then the value of the word as stored in the file is an offset from the associated external symbol. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added into the word in the file.

If relocation information is present, it amounts to one word per word of program text or initialized data. There is no relocation information if the 'relocation info stripped' flag in the header is on.

Bits 3-1 of a relocation word indicate the segment referred to by the text or data word associated with the relocation word:

000 absolute number
002 reference to text segment
004 reference to initialized data
006 reference to uninitialized data (bss)
010 reference to undefined external symbol

If Bit 0 of the relocation word is 1, the reference is relative to the pc (e.g. 'clr x'); if 0, that the reference is to the actual symbol (e.g., 'clr

*$x').

The remainder of the relocation word (bits 15-4) contains a symbol number in the case of external references, and is unused otherwise. The first symbol is numbered 0, the second 1, etc.

SEE ALSO
as, **ld**, **nm** in the optional 8560 Series Native Programming Package.

ar - archive (library) file format

SYNTAX
#include <ar.h>

EXPLANATION
The archive command ar is used to combine several files into one. Archives are used mainly as libraries to be searched by the link-editor ld.

A file produced by ar has a magic number at the start, followed by the constituent files, each preceded by a file header. The magic number and header layout as described in the include file are:

```
#define ARMAG "!<arch>0
#define SARMAG8

#define ARFMAG"`0

struct ar_hdr {
        char  ar_name[16];
        char  ar_date[12];
        char  ar_uid[6];
        char  ar_gid[6];
        char  ar_mode[8];
        char  ar_size[10];
        char  ar_fmag[2];
};
```

The name is a null-terminated string; the date is in the form of **time(2)**; the user ID and group ID are numbers; the mode is a bit pattern per **chmod(2)**; the size is counted in bytes.

Each file begins on a word boundary; a null byte is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

Notice there is no provision for empty areas in an archive file.

SEE ALSO
ar, ld, nm in the optional 8560 Series Native Programming Package.

NOTES
Coding user and group IDs as characters is a bug.

/etc/banproto1 /etc/banproto2 - spooler banner page prototype files

EXPLANATION

Before printing a file on a line printer, the printer spooler prints a banner page, used to separate printouts and to identify the owner of the printout. The format of this banner page is controlled by the file /etc/banprotoN, where N is the number of the associated printer (1 or 2).

The prototype file contains text to be literally copied to the banner page, line formatting characters, and string substitution characters.

The only line formatting character is the up-arrow (^).

^ causes the remainder of the line (excluding any other line formatting characters) to be printed in large type on the banner page.

String substitution characters can appear anywhere in a line and are replaced in the banner page by the corresponding string:

\nnn A backslash immediately followed by one to three octal digits is replaced by the corresponding ASCII character. For example, \014 prints a formfeed character on the banner.

\u is replaced by the username of the user who spooled the file to be printed.

\g is replaced by the groupname of the user who spooled the file to be printed.

\d is replaced by the current date, as output by the **ctime(3)** subroutine.

\newline
A backslash at the end of a line causes the end-of-line character to be ignored. This is used to break a long prototype file line into several shorter ones.

\^ is replaced by an uparrow character (^). This is used to escape the interpretation of the uparrow character.

\\ is replaced by a backslash (\).

\p(fieldspec...)
is replaced by a field from the entry in /etc/passwd associated with the user who spooled the file to be printed.

Fieldspec consists of a field delimiter character (e.g. a colon) immediately followed by one decimal digit. The delimiter character is the character that delimits fields of the line or field; The decimal digit selects one field from that line or field-0 selects the first field.

For example, a banner prototype file might contain the following line:

```
printed by \p(:4)
```

The fieldspec ":4" is read as "field[4] of the password file line". The fifth field (field[4]) of the password file is the comment field.

Some installations further divide the password comment field into subfields separated by semicolons. One such comment field is shown below:

```
Alfred Fetucchini;IH-1113;60-300
```

The first field is the user's real name; the second, his telephone extension; the third, his mail station. A banner prototype line to print his mail station would look like:

```
mail station \p(:4;2)
```

The fieldspec ":4" specifies the password file comment field, just as it did in the previous example. The fieldspec ";2" specifies the third semicolon-separated field of the password comment field (i.e. field[2], Alfred's mail station).

checklist - list of normally mounted file systems

EXPLANATION
The /etc/checklist file contains, for each typically mounted file system, one line showing of the name of the raw logical device that corresponds to that file system (for example, /dev/rhd0). The /etc/checklist file is used by several programs as a list of default file systems to operate on, and shows how your fixed disk drives are arranged (for systems that use 8503 Disk Expansion Units).

For example, consider the following "/etc/checklist" file:

```
/dev/rhd0
/dev/rhd1
/dev/rhd23
```

This file describes one file system located on fixed disk drive 0, a second file system located on fixed disk drive 1, and a third file system spanning fixed disk drives 2 and 3.

FILES
/etc/checklist

NOTES
Be sure to update the /etc/checklist file (to reflect changes to the organization of your 8560's file systems) after installing any 8503 Disk Expansion Units or reconfiguring the 8560's file systems.

core - format of core image file

EXPLANATION
TNIX writes out a core image of a terminated process when any of various errors occur. See **signal**(2) for the list of reasons; the most common are memory violations, illegal instructions, bus errors, and user-generated quit signals. The core image is called 'core' and is written in the process's working directory (provided it can be; normal access controls apply).

The first 1024 bytes of the core image are a copy of the system's per-user data for the process, including the registers as they were at the time of the fault. (See the system listings for the format of this area.) The remainder represents the actual contents of the user's core area when the core image was written. If the text segment is write-protected and shared, it is not dumped; otherwise the entire address space is dumped.

In general the debugger **adb** is sufficient to deal with core images. The **adb** command is included in the optional 8560 Series Native Programming Package.

SEE ALSO
signal(2)
The **adb** command in the optional 8560 Series Native Programming Package.

cvt - structure of kernel CVT table

SYNTAX
#include <sys/cvt.h>

EXPLANATION
The **cvt** data structure in the TNIX kernel contains the commonly-referenced, changeable parameters of the kernel. This allows you to reconfigure the way that the kernel uses its data space, without recompiling the kernel.

A pointer to the **cvt** structure is located at kernel address CVT_LOC. This value is defined in the <sys/cvt.h> include file. The pointer to the actual location of the **cvt** structure is the first address of the structure located at CVT_LOC.

Here is a listing of the <sys/cvt.h> include file:

```
#define CVT_LOC 050
struct cvt_ptr {                /* this is at location CVT_LOC */
        unsigned cp_cvt;        /* pointer to real CVT */
        unsigned cp_kend;       /* end of kernel instruction/data memory */
        unsigned cp_end;        /* end of used memory */
};

struct cvt {                    /* this is at location CVT_LOC->cp_cvt */
        unsigned        c_nbuf;         /* number of buffers */
        unsigned        c_sbuf;         /* size of a buffer entry */
        struct buf      *c_buf;         /* start of buf table */
        struct buf      *c_ebuf;        /* end of buf table */

        char            *c_bstart;      /* start of actual buffers */
        unsigned        c_iocache;      /* number of i/o page buffers */
        unsigned        c_ninode;       /* number of inodes */
        unsigned        c_sinode;       /* size of an inode entry */
        struct inode    *c_inode;       /* start of inode table */
        struct inode    *c_einode;      /* end of inode table */

        unsigned        c_nfile;        /* number of file table entries */
        unsigned        c_sfile;        /* size of a file table entry */
        struct file     *c_file;        /* start of file table */
        struct file     *c_efile;       /* end of file table */

        unsigned        c_nproc;        /* number of process table entries */
        unsigned        c_sproc;        /* size of a process table entry */
        struct proc     *c_proc;        /* start of proc table */
        struct proc     *c_eproc;       /* end of proc table */

        unsigned        c_nmount;       /* number of mountable file systems */
        unsigned        c_smount;       /* size of a mount table entry */
        struct mount    *c_mount;       /* start of mount table */
        struct mount    *c_emount;      /* end of mount table */

        unsigned        c_ntext;        /* number of text table entries */
        unsigned        c_stext;        /* size of a text table entry */
```

```
        struct text     *c_text;        /* start of text table */
        struct text     *c_etext;       /* end of text table */

        unsigned        c_nsmap;        /* number of swapmap table entries */
        unsigned        c_ssmap;        /* size of a swapmap table entry */
        struct map      *c_smap;        /* start of swapmap table */
        struct map      *c_esmap;       /* end of swapmap table */

        unsigned        c_ncmap;        /* number of coremap table entries */
        unsigned        c_scmap;        /* size of a coremap table entry */
        struct map      *c_cmap;        /* start of coremap table */
        struct map      *c_ecmap;       /* end of coremap table */

        dev_t           rootdev;        /* root device (major/minor) */
        dev_t           swapdev;        /* swap device (major/minor) */
        dev_t           pipedev;        /* pipe device (major/minor) */

        unsigned        nswap;          /* amount of swap space */
        daddr_t         swplo;          /* location of swap space */

        time_t          c_time;         /* system time of day */
        char            tz_name[4];     /* time zone name (3 char plus null)*/
        char            tz_dayname[4];  /* daylight time zone name */
        short           tz_offgmt;      /* minutes west of GMT */
        short           tz_flag;        /* time zone flags */
                                        /* bit 0=1; daylight time in effect */
                                        /* (US rules) */

        unsigned        maxmem;         /* max. user process memory (clicks)*/
        unsigned        mem_start;      /* start of user memory (clicks) */
        unsigned        mem_end;        /* end of user memory (clicks) */

        char    *c_pmsg;        /* pointer into buffer */
        char    *c_msgbuf;      /* start of msg buffer */
        char    *c_emsgbuf;     /* end of msg buffer */

        struct IO_Info {        /* buffer cache profiling (bio.c) */
                long    nread;  /* # blocks read */
                long    nreada; /* # blocks read ahead */
                long    ncache; /* # blocks found in cache */
                long    nwrite; /* # blocks written */
                long    *bufcount; /* # free cache blocks (histogram) */
        } io_info;
};
```

FILES

tnix -- a copy of <cvt.h> is located here.
/dev/kmem -- a copy of <cvt.h> is located here.

SEE ALSO

cvt(1)

dir - format of directories

SYNTAX
#include <sys/dir.h>

EXPLANATION
A directory behaves exactly like an ordinary file, except that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its i-node entry. (see **filsys(5).**) The structure of a directory entry as given in the include file is:

```
/*      dir.h   4.2     81/02/19        */

#ifndef DIRSIZ
#define DIRSIZ  14
#endif
struct  direct
{
        ino_t   d_ino;
        char    d_name[DIRSIZ];
};
```

By convention, the first two entries in each directory are "." for the directory itself and ".." for the parent directory. The meaning of ".." is modified for the root directory of the master file system and for the root directories of removable file systems. In the first case, there is no parent, and in the second, the system does not permit off-device references. Therefore in both cases ".." has the same meaning as ".".

SEE ALSO
filsys(5)

dist - format of a distribution floppy

EXPLANATION

A distribution floppy is composed of files stored in fbr(5) format and intended to be accessed using **install(1)** . Each floppy contains a complete subset of the 8560 command repertoire less a (possibly empty) base set of files.

Each distribution has a dependency list which defines the mapping between commands and files. This list is composed of lines of blank separated 'command name', 'file name' pairs. Also, each distribution has two (possibly empty) shell procedure files to handle special case command installations and the running of appropriate installation validation tests.

Files on the floppy with absolute pathnames are conventionally part of the distribution, whereas relative pathnames generally refer to files which exist to support the installation procedure.

FILES

deplist -- dependency list
special -- special case installations
test -- installation validation procedure

SEE ALSO

fbr(1), install(1), fbr(5)

dump, ddate - incremental dump format

SYNTAX

```
#include <sys/types.h>
#include <sys/ino.h>
#include <dumprestor.h>
```

EXPLANATION

Flexible disks used by **dump(1)** and **restor(1)** contain:

- a header record,
- two groups of bit map records,
- a group of records describing directories, and
- a group of records describing files.

Here is the format of the header record and of the first record of each description, as given in the <dumprestor.h> include file:

```
#define NTREC           5
#define MLEN            16
#define MSIZ            4096

#define TS_TAPE         1
#define TS_INODE        2
#define TS_BITS         3
#define TS_ADDR         4
#define TS_END          5
#define TS_CLRI         6
#define MAGIC           (int)60011
#define CHECKSUM        (int)84446

struct  spcl {
        int     c_type;
        time_t  c_date;
        time_t  c_ddate;
        int     c_volume;
        daddr_t c_tapea;
        ino_t   c_inumber;
        int     c_magic;
        int     c_checksum;
        struct  dinode  c_dinode;
        int     c_count;
        char    c_addr[BSIZE];
} spcl;

struct  idates {
        char    id_name[16];
        char    id_incno;
        time_t  id_ddate;
};
```

```
#define DUMPOUTFMT      "%-16s %c %s"           /* for printf */
                                                /* name, incno, ctime(date) */
#define DUMPINFMT       "%16s %c %[^00      /* inverse for scanf */
```

DEFINITIONS

Here are the definitions of the symbolic constants defined with the "# define" statement:

NTREC the number of 512-byte records in a physical tape block.

MLEN the number of bits in a bit map word.

MSIZ the number of bit map words.

The TS_ entries are used in the c_type field to indicate what sort of header this is. The data types and their definitions are:

TS_TAPE tape volume label

TS_INODE a file or directory follows. The c_dinode field is a copy of the disk inode and contains bits telling what sort of file this is.

TS_BITS a bit map follows. This bit map has a one bit for each inode that was dumped.

TS_ADDR a subrecord of a file description. See c_addr, below.

TS_END end of tape record.

TS_CLRI a bit map follows. This bit map contains a "0" bit for each inode that was empty on the file system when dumped.

MAGIC all header records have this number in c_magic.

CHECKSUM header records checksum to this value.

The fields of the header structure are as follows:

c_type the type of the header.

c_date the date the dump was taken.

c_ddate the date the file system was dumped from.

c_volume the current volume number of the dump.

c_tapea the current number of this 512-byte record.

c_inumber the number of the inode being dumped if this is of type TS_INODE.

c_magic this contains the value MAGIC above, truncated as needed.

c_checksum
this contains whatever value is needed to make the record sum to CHECKSUM.

c_dinode this is a copy of the inode as it appears on the file system; see **filsys(5)**.

c_count the count of characters in c_addr.

c_addr an array of characters describing the blocks of the dumped file. A character is non-zero if the block associated with that character was present on the file system; otherwise, the character is 0. If the block was not present on the file system, no block was dumped; the block will be restored as a hole in the file. If there is not sufficient space in this record to describe all of the blocks in a file, TS_ADDR records will be scattered through the file, each one picking up where the last left off.

Each volume except the last ends with a tapemark (read as an end of file). The last volume ends with a TS_END record and then the tapemark.

The structure idates describes an entry of the file /etc/ddate where dump history is kept. The fields of the structure are:

id_name the dumped file system is '/dev/id_nam'.

id_incno the level number of the dump tape; see **dump(1)**.

id_ddate the date of the incremental dump in system format;

FILES
/etc/ddate

SEE ALSO
dump(1), dumpdir(1), filsys(5), restor(1)

environ - user environment

SYNTAX
extern char **environ;

EXPLANATION
An array of strings called the 'environment' is made available by **exec**(2) when a process begins. By convention these strings have the form 'name=value'. The following names are set automatically and are used by various commands:

PATH The sequence of directory prefixes that **sh** applies in searching for a file known by an incomplete path name. The prefixes are separated by ':'. **Login(1)** sets PATH=:/bin:/usr/bin.

HOME A user's login directory, set by **login**(1) from the password file **passwd**(5).

PS1 Shell initial prompt string.

PS2 Shell secondary prompt string.

IFS Shell field separators.

Further names may be placed in the environment by the **export** command and 'name=value' arguments in **sh**(1), or by **exec**(2). It is unwise to conflict with certain shell variables that are frequently exported by '.profile' files:

TERM The kind of terminal for which output is to be prepared.

IU The 8540 associated with this user.

uP The currently selected target microprocessor.

MAIL The pathname of incoming mail. Used by **sh**(1).

SEE ALSO
exec(2), **sh**(1), **login**(1)

fbr - file backup and restore format

SYNTAX

```
#include <sys/types.h>
#include <fbr.h>
```

EXPLANATION

Fbr saves and restores directories and files on a floppy disk archive, preserving aliases (multiple links to the same file). The archive consists of:

Boot block - Block zero is reserved for a copy of a stand-alone boot program.

Directory area - This area contains directory information for all files and directories on the archive. The first entry in this area is dedicated to the archive label.

Data area - The data of all files on the archive is stored here.

The format of a directory entry (as given in the include file) follows. To increase the portability of archives, this structure is read and written in PDP-11 format regardless of what machine **fbr** is running on.

```
#define PATHLEN         106     /* space for path       */

struct fbrent
{
        char            fbr_path[PATHLEN]; /* pathname  */
        unsigned short  fbr_mode;
        short           fbr_uid;  /* owner's userID     */
        short           fbr_gid;  /* owner's groupID    */
        off_t           fbr_size; /* size in bytes      */
        time_t          fbr_acct; /* access date-time   */
        time_t          fbr_modt; /* modify date-time   */
        unsigned        fbr_fblk; /* first data block   */
        char            fbr_zero; /* unused             */
        char            fbr_chks; /* checksum           */
};
```

The path field is the pathname of the file when archived, less any redundant slashes or initial './'. It is null-terminated if less than PATHLEN bytes long. The mode, uid, gid, size, and access and modification date-times are in the same format as their corresponding i-node fields. As in the file system, an available entry has a mode of zero. The fblk field contains the block number of the first data block allocated to this file. The zero field is unused. The checksum field contains a number such that the sum of the first 127 bytes of the directory entry and the complement of the checksum is zero. Each file's data starts on a block boundary and occupies max(1,((size + 511) / 512) contiguous blocks. At least one archive block is allocated to each archived file or directory so that aliases (multiple links) can be properly recorded. Each directory archived is treated as an empty file

(i.e., no directory's data is stored). All entries representing links to a given file are identical. In particular, the fblk field in each link's entry contains the same block number.

The archive label directory entry contains the following information:

path A comment about this archive.

mode Set by **fbr** to a file readable/writeable/executable by all. Otherwise unused.

uid The userID of the user who created this archive.

gid The groupID of the same.

size The total number of bytes in the data area of the archive. This field in combination with the **fblk** field, records the size of both the directory area and the entire archive.

acct The date-time that this archive was created - not its access time.

modt The date-time that this archive was last altered.

fblk The block number of the first archive block following the directory area.

checksum
(same as any other directory entry.)

SEE ALSO
fbr(1), stat(2)

filsys, flblk, ino - format of file system volume

SYNTAX

```
#include <sys/types.h>
#include <sys/flbk.h>
#include <sys/filsys.h>
#include <sys/ino.h>
```

EXPLANATION

Every file system storage volume (e.g. disk) has a common format for certain vital information. Every such volume is divided into a certain number of 512-byte blocks. Block 0 is unused and is available to contain a bootstrap program, pack label, or other information.

Block 1 is the super block. The layout of the super block as defined by the include file <sys/filsys.h> is:

```
/*
 * Structure of the super-block
 */
struct filsys {
        unsigned short s_isize;     /* size in blocks of i-list */
        daddr_t  s_fsize;           /* size in blocks of entire volume */
        short    s_nfree;           /* number of addresses in s_free */
        daddr_t  s_free[NICFREE];/* free block list */
        short    s_ninode;          /* number of i-nodes in s_inode */
        ino_t    s_inode[NICINOD];/* free i-node list */
        char     s_flock;           /* lock during free list manipulation */
        char     s_ilock;           /* lock during i-list manipulation */
        char     s_fmod;            /* super block modified flag */
        char     s_ronly;           /* mounted read-only flag */
        time_t   s_time;            /* last super block update */
};
```

S_isize is the address of the first block after the i-list, which starts just after the super-block, in block 2. Thus i-list is **s_isize**-2 blocks long. **S_fsize** is the address of the first block not potentially available for allocation to a file. These numbers are used by the system to check for bad block addresses; if an 'impossible' block address is allocated from the free list or is freed, a diagnostic is written on the on-line console. Moreover, the free array is cleared, so as to prevent further allocation from a presumably corrupted free list.

The free list for each volume is maintained as follows. The **s_free** array contains, in **s_free[1], ... , s_free[s_nfree-1]**, up to NICFREE free block numbers. NICFREE is a configuration constant. **S_free[0]** is the block address of the head of a chain of blocks constituting the free list. The layout of each block of the free chain as defined in the include file <sys/fblk.h> is:

```
struct fblk
{
        int     df_nfree;
        daddr_t df_free[NICFREE];
```

};

The fields **df_nfree** and **df_free** in a free block are used exactly like **s_nfree** and **s_free** in the super block. To allocate a block: decrement **s_nfree**, and the new block number is **s_free[s_nfree]**. If the new block address is 0, there are no blocks left, so give an error. If **s_nfree** became 0, read the new block into **s_nfree** and **s_free**. To free a block, check if **s_nfree** is NICFREE; if so, copy **s_nfree** and the **s_free** array into it, write it out, and set **s_nfree** to 0. In any event set **s_free[s_nfree]** to the freed block's address and increment **s_nfree**.

S_ninode is the number of free i-numbers in the **s_inode** array. To allocate an i-node: if **s_ninode** is greater than 0, decrement it and return **s_inode[s_ninode]**. If it was 0, read the i-list and place the numbers of all free inodes (up to NICINOD) into the **s_inode** array, then try again. To free an i-node, provided **s_ninode** is less than NICINODE, place its number into **s_inode[s_ninode]** and increment **s_ninode**. If **s_ninode** is already NICINODE, don't bother to enter the freed i-node into any table. This list of i-nodes is only to speed up the allocation process; the information as to whether the inode is really free or not is maintained in the inode itself.

S_flock and **s_ilock** are flags maintained in the memory copy of the file system while it is mounted and their values on disk are immaterial. The value of **s_fmod** on disk is likewise immaterial; it is used as a flag to indicate that the super-block has changed and should be copied to the disk during the next periodic update of file system information. **S_ronly** is a write-protection indicator; its disk value is also immaterial.

S_time is the last time the super-block of the file system was changed. During a reboot, **s_time** of the super-block for the root file system is used to set the system's idea of the time.

The fields **s_tfree**, **s_tinode**, **s_fname**, **s_m**, **s_n**, and **s_fpack** are not currently maintained.

I-numbers begin at 1, and the storage for i-nodes begins in block 2. I-nodes are 64 bytes long, so 8 of them fit into a block. I-node 1 is used to group bad blocks found on the disk. I-node 2 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each i-node represents one file. The format of an i-node as given in the include file **<sys/ino.h>** is:

```
/*
 * Inode structure as it appears on
 * a disk block.
 */
struct dinode
{
        unsigned short  di_mode;     /* mode and type of file */
        short   di_nlink;    /* number of links to file */
        short   di_uid;      /* owner's user id */
        short   di_gid;      /* owner's group id */
        off_t   di_size;     /* number of bytes in file */
        char    di_addr[40]; /* disk block addresses */
```

```
        time_t  di_atime;       /* time last accessed */
        time_t  di_mtime;       /* time last modified */
        time_t  di_ctime;       /* time created */
};
#define INOPB   8       /* 8 inodes per block */
/*
 * the 40 address bytes:
 *      39 used; 13 addresses
 *      of 3 bytes each.
 */
```

Di_mode tells the kind of file; it is encoded identically to the **st_mode** field of **stat(2)**. **Di_nlink** is the number of directory entries (links) that refer to this i-node. **Di_uid** and **di_gid** are the owner's user and group IDs. **Size** is the number of bytes in the file. **Di_atime** and **di_mtime** are the times of last access and modification of the file contents (read, write or create). (See **times(2).**) **Di_ctime** records the time of last modification to the inode or to the file, and is used to determine whether it should be dumped.

Special files are recognized by their modes and not by i-number. A block-type special file is one which can potentially be mounted as a file system; a character-type special file cannot, though it is not necessarily character-oriented. For special files, the **di_addr** field is occupied by the device code. The device codes of block and character special files overlap.

Disk addresses of plain files and directories are kept in the array **di_addr** packed into 3 bytes each. The first 10 addresses specify device blocks directly. The last 3 addresses are singly, doubly, and triply indirect and point to blocks of 128 block pointers. Pointers in indirect blocks have the type **daddr_t**.

For block **b** in a file to exist, it is not necessary that all blocks less than **b** exist. A zero block number either in the address words of the i-node or in an indirect block indicates that the corresponding block has never been allocated. Such a missing block reads as if it contained all zero words.

SEE ALSO
syschk(1), dir(5), stat(2)

gpib - GPIB Interface data files

EXPLANATION
Two files contain data for the GPIB Interface:

- /etc/gpib/dev contains information about each GPIB-compatible device that exists on the system.
- /etc/gpib/gd maps GD names (device driver routine names) into identification numbers.

Both files are ASCII text. The **mkgpib(1)** command maintains the files.

There is a one-line entry in /etc/gpib/dev for each GPIB-compatible device. Each entry has seven text fields that are separated by colons. The fields contain the following information:

1. The device's complete pathname.
2. The device's minor device number.
3. The name of the device's GD.
4. The device's GPIB primary address.
5. The device's GPIB secondary address.
6. A comment that describes the device. The comment may be up to 80 characters long.
7. One or more device-dependent options (separated by ";").

There is a one-line entry in /etc/gpib/gd for each GD name. Each entry has two fields that are separated by a colon. The first field contains the GD name, and the second field contains the GD's identification number.

SEE ALSO
gpib(4), mkgpib(1)

group - group file

EXPLANATION
Group contains for each group the following information:

- group name
- encrypted password
- numerical group ID
- a comma separated list of all users allowed in the group

Group is an ASCII file. The fields are separated by colons (:). Each group is separated from the next by a newline. If the password field is null, no password is demanded.

This file resides in directory /etc. Because of the encrypted passwords, group has general read permission and can be used, for example, to map numerical group IDs to names.

FILES
/etc/group

SEE ALSO
newgrp(1), crypt(3), passwd(1), passwd(5)

mtab - mounted file system table

EXPLANATION

/etc/mtab is a table of currently mounted file systems. The **mount** command adds entries to this table; **umount** removes them.

Each entry is 64 bytes long. The first 32 are the null-padded name of the file the file system is mounted on (e.g. "/usr1"); the last 32 are the null-padded name of the special file involved (less any directory names: /dev/hd0 would be stored as "hd0").

FILES
/etc/mtab

SEE ALSO
mount(1), mount(2)

passwd - password file

EXPLANATION
Passwd contains for each user the following information:

login name
encrypted password
numerical user ID
numerical group ID
a spare field for future use
initial working directory
program to use as Shell

This is an ASCII file. Each field within each user's entry is separated from the next by a colon. The spare field can contain any desired information. Each user is separated from the next by a new-line. If the password field is null, no password is demanded; if the Shell field is null, the Shell itself is used.

This file resides in directory /etc. Because of the encrypted passwords, passwd has general read permission and can be used, for example, to map numerical user ID's to names.

FILES
/etc/passwd

SEE ALSO
getpwent(3), login(1), crypt(3), passwd(1), group(5)

plot - graphics interface

EXPLANATION

Files of this format are produced by routines described in **plot(3)**, and are interpreted for various devices by commands described in **plot**. This **plot** command is included in the optional 8560 Series Auxiliary Utilities Package.

A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order. A point is designated by four bytes representing the x and y values; each value is a signed integer. The last designated point in an **l**, **m**, **n**, or **p** instruction becomes the 'current point' for the next instruction.

Each of the following descriptions begins with the name of the corresponding routine in **plot(3)**.

m move: The next four bytes give a new current point.

n cont: Draw a line from the current point to the point given by the next four bytes.

p point: Plot the point given by the next four bytes.

l line: Draw a line from the point given by the next four bytes to the point given by the following four bytes.

t label: Place the following ASCII string so that its first character falls on the current point. The string is terminated by a newline.

a arc: The first four bytes give the center, the next four give the starting point, and the last four give the end point of a circular arc. The least significant coordinate of the end point is used only to determine the quadrant. The arc is drawn counter-clockwise.

c circle: The first four bytes give the center of the circle, the next two the radius.

e erase: Start another frame of output.

f linemod: Take the following string, up to a newline, as the style for drawing further lines. The styles are 'dotted,' 'solid,' 'longdashed,' 'shortdashed,' and 'dotdashed.' Effective only in plot 4014 and plot ver.

s space: The next four bytes give the lower left corner of the plotting area; the following four give the upper right corner. The plot will be magnified or reduced to fit the device as closely as possible.

Space settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of **plot**. The upper limit is just outside the plotting area. In every case the plotting area is taken to be square; points outside may be displayable on devices whose face isn't square.

```
4014 space(0, 0, 3120, 3120);
ver  space(0, 0, 2048, 2048);
300, 300s
     space(0, 0, 4096, 4096);
450  space(0, 0, 4096, 4096);
```

SEE ALSO
plot(3), graph(1)
plot in the optional 8560 Series Auxiliary Utilities Package.

termcap - terminal capability data base

SYNTAX
/etc/termcap

EXPLANATION
Termcap(*) is a data base describing terminals, used, e.g., by **ksh** and **ace**. Terminals are described in termcap by giving their capabilities and by describing how they perform certain operations. Padding requirements and initialization sequences are included in termcap.

Entries in termcap consist of a number of ':' separated fields. The first entry for each terminal gives the names which are known for the terminal, separated by '|' characters. The first name is always 2 characters long and is used by older version 6 systems which store the terminal type in a 16 bit word in a systemwide data base. The second name given is the most common abbreviation for the terminal, and the last name given should be a long name fully identifying the terminal. The second name should contain no blanks; the last name may well contain blanks for readability.

CAPABILITIES
(P) indicates padding may be specified
(P*) indicates that padding may be based on no. lines affected

Name	Type	Pad?	Description
ae	str	(P)	End alternate character set
al	str	(P*)	Add new blank line
am	bool		Terminal has automatic margins
as	str	(P)	Start alternate character set
BC	str		Sets character background color on color terminal. Ignored unless FC and WC are also set.
bc	str		Backspace if not ^H
bs	bool		Terminal can backspace with ^H
bt	str	(P)	Back tab
bw	bool		Backspace wraps from column 0 to last column
CC	str		Command character in prototype if terminal settable
cd	str	(P*)	Clear to end of display
ce	str	(P)	Clear to end of line
ch	str	(P)	Like cm but horizontal motion only, line stays same
cl	str	(P*)	Clear screen
cm	str	(P)	Cursor motion
co	num		Number of columns in a line
cr	str	(P*)	Carriage return, (default ^M)
cs	str	(P)	Change scrolling region (vt100), like cm
cv	str	(P)	Like ch but vertical only.
da	bool		Display may be retained above
dB	num		Number of millisec of bs delay needed
db	bool		Display may be retained below
dC	num		Number of millisec of cr delay needed
dc	str	(P*)	Delete character
dF	num		Number of millisec of ff delay needed

(*) The **termcap** database has been provided by the Computer Science Department of the University of California at Berkeley.

dl	str	(P*)	Delete line
dm	str		Delete mode (enter)
dN	num		Number of millisec of nl delay needed
do	str		Down one line
dT	num		Number of millisec of tab delay needed
ed	str		End delete mode
ei	str		End insert mode; give :ei=: if **ic**
eo	str		Can erase overstrikes with a blank
FC	str		Sets foreground color on color terminals. Ignored unless BC and WC are also set.
ff	str	(P*)	Hardcopy terminal page eject (default ^L)
hc	bool		Hardcopy terminal
hd	str		Half-line down (forward 1/2 linefeed)
ho	str		Home cursor (if no **cm**)
hu	str		Half-line up (reverse 1/2 linefeed)
hz	str		Hazeltine; can't print ~'s
ic	str	(P)	Insert character
if	str		Name of file containing **is**
im	bool		Insert mode (enter); give :im=: if **ic**
in	bool		Insert mode distinguishes nulls on display
ip	str	(P*)	Insert pad after character inserted
is	str		Terminal initialization string
KA	str		Sent by function key 11.
KB	str		Sent by function key 12.
KC	str		Sent by function key 13.
KD	str		Sent by function key 14.
KE	str		Sent by function key 15.
KF	str		Sent by function key 16.
k0-k9	str		Sent by other function keys 0-9
kb	str		Sent by backspace key
kd	str		Sent by terminal down arrow key
ke	str		Out of keypad transmit mode
kh	str		Sent by home key
kl	str		Sent by terminal left arrow key
kn	num		Number of other keys
ko	str		Termcap entries for other non-function keys
kr	str		Sent by terminal right arrow key
ks	str		Put terminal in keypad transmit mode
ku	str		Sent by terminal up arrow key
l0-l9	str		Labels on other function keys
li	num		Number of lines on screen or page
ll	str		Last line, first column (if no **cm**)
ma	str		Arrow key map
mi	bool		Safe to move while in insert mode
ml	str		Memory lock on above cursor.
ms	bool		Safe to move while in standout and underline mode
mu	str		Memory unlock (turn off memory lock).
nc	bool		No correctly working carriage return (DM2500,H2000)
nd	str		Non-destructive space (cursor right)
nl	str	(P*)	Newline character (default **\n**)
ns	bool		Terminal is a CRT but doesn't scroll.
os	bool		Terminal overstrikes
pc	str		Pad character (rather than null)
pt	bool		Has hardware tabs (may need to be set with **is**)

se	str		End stand out mode
sf	str	(P)	Scroll forwards
sg	num		Number of blank chars left by so or se
so	str		Begin stand out mode
sr	str	(P)	Scroll reverse (backwards)
ta	str	(P)	Tab (other than ^I or with padding)
tc	str		Entry of similar terminal - must be last
te	str		String to end programs that use cm
ti	str		String to begin programs that use cm
uc	str		Underscore one char and move past it
ue	str		End underscore mode
ug	num		Number of blank chars left by us or ue
ul	bool		Terminal underlines even though it doesn't overstrike
up	str		Upline (cursor up)
us	str		Start underscore mode
vb	str		Visible bell (may not move cursor)
ve	str		Sequence to end open/visual mode
vs	str		Sequence to start open/visual mode
WC	str		Sets wash color on color terminals. Ignored unless FC and BC are set.
YK	bool		Setting puts ksh labels on upper screen.
xb	bool		Beehive (f1=escape, f2=ctrl C)
xn	bool		A newline is ignored after a wrap (Concept)
xr	bool		Return acts like ce \r \n (Delta Data)
xs	bool		Standout not erased by writing over it (HP 264?)
xt	bool		Tabs are destructive, magic so char (Teleray 1061)

EXAMPLES

The following entry, which describes the Concept-100, is among the more complex entries in the termcap file. (This particular concept entry is outdated, and is used as an example only.)

```
c1|c100|concept100:is=\EU\Ef\E7\E5\E8\E1\ENH\EK\E\200\Eo&\200:\
        :al=3*\E^R:am:bs:cd=16*\E^C:ce=16\E^S:cl=2*^L:cm=\Ea%+ %+ :co#80:\
        :dc=16\E^A:dl=3*\E^B:ei=\E\200:eo:im=\E^P:in:ip=16*:li#24:mi:nd=\E=:\
        :se=\Ed\Ee:so=\ED\EE:ta=8\t:ul:up=\E;:vb=\Ek\EK:xn:
```

Entries may continue onto multiple lines by giving a \ as the last character of a line, and empty fields may be included for readability (here between the last field on a line and the first field on the next). Capabilities in termcap are of three types: Boolean capabilities which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities, which give a sequence which can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have two-letter codes. For instance, the fact that the Concept has automatic margins (i.e. an automatic return and linefeed when the end of a line is reached) is indicated by the capability **am**. Hence the

description of the Concept includes **am**. Numeric capabilities are followed by the character '#' and then the value. Thus **co** which indicates the number of columns the terminal has gives the value '80' for the Concept.

Finally, string valued capabilities, such as **ce** (clear to end of line sequence) are given by the two-character code, an '=', and then a string ending at the next following ':'. A delay in milliseconds may appear after the '=' in such a capability, and padding characters are supplied by the editor after the remainder of the string is sent to provide this delay. The delay can be either a integer, e.g. '20', or an integer followed by an '*', i.e. '3*'. A '*' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. When a '*' is specified, it is sometimes useful to give a delay of the form '3.5' specify a delay per unit to tenths of milliseconds.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. A **\E** maps to an ESCAPE character, **^x** maps to a control-x for any appropriate x, and the sequences **\n \r \t \b \f** give a newline, return, tab, backspace and formfeed. Finally, characters may be given as three octal digits after a \, and the characters ^ and \ may be given as \^ and \\. If it is necessary to place a : in a capability it must be escaped in octal as **\072**. If it is necessary to place a null character in a string capability it must be encoded as **\200**. The routines which deal with termcap use C strings, and strip the high bits of the output very late so that a **\200** comes out as a **\000** would.

Preparing Descriptions

We now outline how to prepare descriptions of terminals. The most effective way to prepare a terminal description is by imitating the description of a similar terminal in termcap and to build up a description gradually, using partial descriptions to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the termcap file to describe it. To test a new terminal description easily, you can set the environment variable TERMCAP to a pathname of a file containing the description you are working on and the editor will look there rather than in /etc/termcap. TERMCAP can also be set to the termcap entry itself to avoid reading the file when starting up the editor.

Basic capabilities

The number of columns on each line for the terminal is given by the **co** numeric capability. If the terminal is a CRT, then the number of lines on the screen is given by the **li** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, then this is given by the **cl** string capability. If the terminal can backspace, then it should have the **bs** capability, unless a backspace is accomplished by a character other than **^H**, in which case you should give this character as the **bc** string capability. If it overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability.

A very important point here is that the local cursor motions encoded in termcap are undefined at the left and top edges of a CRT terminal. The editor will never attempt to backspace around the left edge, nor will it attempt to go up locally off the top. The editor assumes that feeding off the bottom of the screen will cause the screen to scroll up, and the **am** capability tells whether the cursor sticks at the right edge of the screen. If the terminal has switch selectable automatic margins, the termcap file usually assumes that this is on, i.e. **am**.

These capabilities suffice to describe hardcopy and glass-tty terminals. Thus the model 33 teletype is described as

```
t3|33|tty33:co#72:os
```

while the Lear Siegler ADM-3 is described as

```
cl|adm3|3|lsi adm3:am:bs:cl=^Z:li#24:co#80
```

Cursor addressing

Cursor addressing in the terminal is described by a **cm** string capability containing printf-like escapes (**%x**). These substitute to encodings of the current line or column position, while other characters are passed through unchanged. If the **cm** string is thought of as a function, then its arguments are the line and then the column to which motion is desired, and the **%** encodings have the following meanings:

%d	as in printf, 0 origin
%2	like %2d
%3	like %3d
%.	like %c
%+x	adds x to value, then %.
%>xy	if value > x adds y, no output.
%r	reverses order of line and column, no output
%i	increments line/column (for 1 origin)
%%	gives a single %
%n	exclusive or row and column with 0140 (DM2500)
%B	BCD (16*(x/10)) + (x%10), no output.
%D	Reverse coding (x-2*(x%16)), no output. (Delta Data).

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cm** capability is cm=6\E&%r%2c%2Y. The Microterm ACT-IV needs the current row and column sent preceded by a ^T, with the row and column simply encoded in binary, cm=^T%.%.. Terminals which use %. need to be able to backspace the cursor (**bs** or **bc**), and to move the cursor up one line on the screen (**up** introduced below). This is necessary because it is not always safe to transmit \t, \n ^D and \r, as the system may change or discard them.

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus cm=\E=%+ %+ .

Cursor motions

If the terminal can move the cursor one position to the right, leaving the character at the current position unchanged, then this sequence should be given as **nd** (non-destructive space). If it can move the cursor up a line on the screen in the same column, this should be given as **up**. If the terminal has no cursor addressing capability, but can home the cursor (to the very upper left corner of screen) then this can be given as **ho**; similarly a fast way of getting to the lower left corner can be given as **ll**; this may involve going up with **up** from the home position, but the editor will never do this itself (unless **ll** does) because it makes no assumption about the effect of moving up from the home position.

Area clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **ce**. If the terminal can clear from the current position to the end of the display, then this should be given as **cd**. The editor only uses **cd** from the first column of a line.

Insert/delete line

If the terminal can open a new blank line before the line where the cursor is, this should be given as **al**; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as **dl**; this is done only from the first position on the line to be deleted. If the terminal can scroll the screen backwards, then this can be given as **sb**, but just **al** suffices. If the terminal can retain display memory above then the **da** capability should be given; if display memory can be retained below then **db** should be given. These let the editor understand that deleting a line on the screen may bring non-blank lines up from below or that scrolling back with **sb** may bring down non-blank lines.

Insert/delete character

There are two basic kinds of intelligent terminals with respect to insert/delete character which can be described using termcap. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can find out which kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type abc def using local cursor motions (not spaces) between the abc and the def. Then position the cursor before the abc and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the abc shifts

over to the def which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for insert null. If your terminal does something different and unusual then you may have to modify the editor to get it to use the insert mode your terminal defines. Terminals which have an insert mode not falling into one of these two classes are very rare.

The editor can handle both terminals which have an insert mode, and terminals which send a simple sequence to open a blank position on the current line. Give as **im** the sequence to get into insert mode, or give it an empty value if your terminal uses a sequence to insert a blank position. Give as **ei** the sequence to leave insert mode (give this, with an empty value also if you gave **im** so). Now give as **ic** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give **ic**, terminals which send a sequence to open a screen position should give it here. (Insert mode is preferable to the sequence to open a position on the screen if your terminal has both.) If post insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in **ip**.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g. if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability **mi** to speed up inserting in this case. Omitting **mi** will affect only speed. Some terminals (notably Datamedia's) must not have **mi** because of the way their insert mode works.

Finally, you can specify delete mode by giving **dm** and **ed** to enter and exit delete mode, and **dc** to delete a single character while in delete mode.

Highlighting, underlining, and visible bells

If your terminal has sequences to enter and exit standout mode these can be given as **so** and **se** respectively. If there are several flavors of standout mode (such as inverse video, blinking, or underlining - half bright is not usually an acceptable standout mode unless the terminal is in inverse video mode constantly) the preferred mode is inverse video by itself. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then **ug** should be given to tell how many spaces are left.

Codes to begin underlining and end underlining can be given as **us** and **ue** respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as **uc**. (If the underline code does not move the cursor to the right, give the code followed by a nondestructive space.)

Many terminals, such as the HP 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement) then this can be given as **vb**; it must not move the cursor. If the terminal should be placed in a different mode during open and visual modes of an editor, this can be given as **vs** and **ve**, sent at the start and end of these modes respectively. These can be used to change, e.g., from an underline to a block cursor and back.

If the terminal needs to be in a special mode when running a program that addresses the cursor, the codes to enter and exit this mode can be given as **ti** and **te**. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly.

If your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability **ul**. If overstrikes are erasable with a blank, then this should be indicated by giving **eo**.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted HP 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **ks** and **ke**. Otherwise the keypad is assumed to always transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kl, kr, ku, kd,** and **kh** respectively. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as **k0, k1, ..., k9**. If these keys have labels other than the default f0 through f9, the labels can be given as **l0, l1, ..., l9**. If there are other keys that transmit the same code as the terminal expects for the corresponding function, such as clear screen, the termcap two-letter codes can be given in the **ko** capability, for example, :ko=cl,ll,sf,sb:, which says that the terminal has clear, home down, scroll down, and scroll up keys that transmit the same thing as the cl, ll, sf, and sb entries.

The **ma** entry is also used to indicate arrow keys on terminals which have single character arrow keys. It is obsolete but still in use in version 2 of vi (a text editor not supplied with TNIX), which must be run on some minicomputers due to memory limitations. This field is redundant with **kl, kr, ku, kd,** and **kh**. It consists of groups of two characters. In each group, the first character is what an arrow key sends, the second character is the corresponding vi command. These commands are **h** for **kl**, **j** for **kd**, **k** for **ku**, **l** for **kr**, and **H** for **kh**. For example, the mime would be **:ma=^Kj^Zk^Xl:** indicating arrow keys left (^H), down (^K), up (^Z), and right (^X). (There is no home key on the mime.)

Miscellaneous
If the terminal requires other than a null (zero) character as a pad, then this can be given as **pc**.
If tabs on the terminal require padding, or if the terminal uses a character other than ^I to tab, then this can be given as **ta**.

Hazeltine terminals, which don't allow '~' characters to be printed should indicate **hz**. Datamedia terminals, which echo carriage-return linefeed for carriage return and then ignore a following linefeed should indicate **nc**. Early Concept terminals, which ignore a linefeed immediately after an **am** wrap, should indicate **xn**. If an erase-eol is required to get rid of standout (instead of merely writing on top of it), **xs** should be given. Teleray terminals, where tabs turn all characters moved over to blanks, should indicate **xt**. Other specific terminal problems may be corrected by adding more capabilities of the form **xx**.

Other capabilities include **is**, an initialization string for the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to properly clear and then set the tabs on the terminal, if the terminal has settable tabs. If both are given, **is** will be printed before **if**. This is useful where **if** is /usr/lib/tabset/std but **is** clears the tabs first.

Similar Terminals
If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **tc** can be given with the name of the similar terminal. This capability must be last and the combined length of the two entries must not exceed 1024. Since the routines that access termcap search the entry from left to right, and since the tc capability is replaced by the corresponding entry, the capabilities given at the left override the ones in the similar terminal. A capability can be canceled with **xx@** where xx is the capability. For example, the entry

```
hn|2621nl:ks@:ke@:tc=2621:
```

defines a 2621nl that does not have the **ks** or **ke** capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

FILES
/etc/termcap -- file containing terminal descriptions

SEE ALSO
termcap, **tset, more**

NOTES
The total length of a single entry (excluding only escaped newlines) may not exceed 1024.

Not all programs support all entries. There are entries that are not supported by any program.

ttys - terminal initialization data

EXPLANATION
The ttys file is read by the init program and specifies which terminal special files are to have a process created for them which will allow people to log in. It contains one line per special file.

The first character of a line is either '0' or '1'; the former causes the line to be ignored, the latter causes it to be effective. The second character is used as an argument to **getty(1)**, which performs such tasks as baud-rate recognition, reading the login name, and calling **login**. The characters recognized by **getty(1)** are included with the description of **getty**.

FILES
/etc/ttys

SEE ALSO
init(1), getty(1), login(1)

utmp, wtmp - login records

SYNTAX
#include <utmp.h>

EXPLANATION
The utmp file allows one to discover information about who is currently using **TMIX**. The file is a sequence of entries with the following structure declared in the include file:

```
struct utmp {
        char    ut_line[8];             /* tty name */
        char    ut_name[8];             /* user id */
        long    ut_time;                /* time on */
};
```

This structure gives the name of the special file associated with the user's terminal, the user's login name, and the time of the login in the form of **time(2)**.

The wtmp file records all logins and logouts. Its format is exactly like utmp except that a null user name indicates a logout on the associated terminal. Furthermore, the terminal name '~' indicates that the system was rebooted at the indicated time; the adjacent pair of entries with terminal names '|' and '}' indicate the system-maintained time just before and just after a date command has changed the system's idea of the time.

Wtmp is maintained by **login(1)** and **init(1)**. Neither of these programs creates the file, so if it is removed record-keeping is turned off. It is summarized by **ac(1)**.

FILES
/etc/utmp
/usr/adm/wtmp

SEE ALSO
login(1), init(1), who(1)

Section 6

GLOSSARY

Ancestor. See parent.

Argument. A name or expression in a TNIX command line that supplies information to the command, command file, or program to be executed.

Background Execution. The process of executing a list of one or more commands concurrently--without waiting for these commands to finish executing--by placing an ampersand (&) character at the end of a command line.

Bad Block. A block of disk space that is no longer usable due to physical damage. A bad block is not allocated once it has been marked by the TNIX **syschk** command.

Blank Interpretation. The process of parsing a shell command line into its constituent arguments. Variable and command substitutions are performed before the command line arguments are separated into individual arguments. The shell variable IFS defines characters that the shell uses to separate command line arguments.

Block Special File. A software device driver that transfers data to/from an I/O device in units of blocks.

Block. (1) Any 512-byte segment of disk space, memory, data, or program code. (2) In Tekhex protocol, any message up to 255 bytes in length.

Category C. Optional software supplied by Tektronix, for which Tektronix makes no warranty, express or implied, that the software is suitable for a specific purpose or that it performs any specific function correctly. Category C software includes:

- Optional Text Processing Package
- Optional Native Programming Package
- Optional Auxiliary Utilities Package

Character Special File. A software device driver that transfers data character-by-character to or from an I/O device (such as a terminal).

Child. The process created when an executing process creates another concurrently executing process with the **fork** TNIX system call. (The originating process is called the parent process.)

ColorKey+. An interface to TNIX that enables you to enter commands by pressing function keys as well as typing the commands literally.

COM Interface. A communications interface similar to TERM mode, but more restrictive. With the COM interface, communicates with the 8560 primarily for the purpose of transferring object code and files. The COM interface is most useful for communicating with the 8001 and 8002A (which do not support TERM mode). See the Communication with 8540s and 8550s section of 8560 Series System Users Manual for a brief description of the COM interface.

Command File. A file that contains commands to be processed by the TNIX shell or by a system program such as the linker, the library generator, or the ACE editor.

Command Substitution. A feature of the TNIX shell that replaces one or more commands enclosed in accent grave characters (`) with the output from those commands, supplying the output of the accent-grave-enclosed characters as a command line argument to a command.

Control Character. A character whose ASCII code is in the hexadecimal range 00 to 1F, or 7F. Some control characters are entered by pressing special keys, such as TAB, ESC, or RETURN. Others are entered by simultaneously pressing the CTRL key and some other key.

Cooked Device. A software device driver that allows the system to process transferred data.

Current Directory. The directory that you are currently accessing. The current directory is used as a starting place when a full pathname (a name that begins with "/") is not given. The current directory is sometimes referred to as the working directory.

Daemon. An independently executing process that asynchronously performs periodic system maintenance tasks, such as printing files from a spool directory. (A spool directory contains files that are queued, waiting to be printed on the line printer.)

Default. A predefined value for a command parameter, used when no value for the parameter is explicitly specified.

Descendant. File F is termed a descendant of directory D if:

- D contains F, or
- a descendant of D contains F.

Device. (1) A piece of computer hardware. (2) An instrument attached to the 8560 that is used for data entry, storage, and display. For device I/O, TNIX treats devices as special files. Thus, you can direct a program's output to a special file the same way you would direct that program's output to a regular file. All special files are located in the /dev directory. See logical device.

Device Number. See major device number.

Directory. A file that contains only pointers (called links) to other files. A file that is pointed to by a directory is said to reside in the directory; every file resides in at least one directory. Similarly, a directory is said to contain each file that the directory points to. The mkdir command creates a new directory.

DOS/50. The operating system of the 8550 Microcomputer Development System.

Download. To transfer data from a host such as the 8560 to a microcomputer development system (such as the 8540).

Environment. A set of string-valued parameters maintained by the shell. These parameters can be accessed by programs executed by the shell. The environment consists of:

- the current directory,
- your user name,
- your group name,
- the default shell variables HOME, IFS, PATH, PS1, PS2, and
- any shell variables that you have defined and "exported" with the shell's **export** command.

Execute Permission. The ability to execute a file or search a directory. Permission is regulated by mode bits, which can be set with the **chmod** command.

File. The fundamental unit of data storage used by the TNIX operating system. A file is a collection of logically related information that is stored on a backup tape, a flexible disk, or a fixed disk drive. TNIX does not assign any specific structure to a file.

File Descriptor. An integer in the range 0--20 that is returned by a **creat**, **dup**, **open**, or **pipe** TNIX system call. The file descriptor, instead of a filename, is used as a parameter to TNIX system calls that access open files. These TNIX system calls include **close**, **ioctl**, **lseek**, **read**, **fstat**, and **write**. The file descriptors 0, 1, and 2 are assigned to standard input, standard output, and standard error, respectively.

File Tree. A directory and all its descendants.

Filename. A sequence of 1 to 14 characters that specifies a file. The "/" and the null character cannot be used in a filename. You should also avoid using control characters, blank, newline, tab, or the following special characters in filenames:

```
&  \  '  `  "  ;  :  ?  !  -  $  #  <  >  [  ]  (  )
```

Filesystem. A complete directory hierarchy that may extend over one or more fixed disk drives. The "root" filesystem is the "/" filesystem--other filesystems are mounted on directories within the root filesystem. Each filesystem has a maximum number of files that may be created on it. Files may not be linked across filesystems.

Filter. A program such as **grep** or **tr** that accepts data from standard input, performs some transformation upon the data, and writes the data to standard output.

Fork. A system call that creates a new (child) process that executes concurrently with the calling, or parent, process. The child process inherits the open files of the parent and executes concurrently with the parent. To avoid concurrent execution of the child and parent process, the parent process can execute the **wait** TNIX system call, which causes it to pause until the child process terminates.

Group. Groups provide a simple method for restricting read/write/execute privileges for specific collections of files and directories to a subset of the users on a specific machine. Each group is assigned a specific entry in the /etc/group file; users who have access to that group are listed in that group's entry in the /etc/group file.

Group ID. The numerical identification of a user as a member of a group of users working on related information.

Groupname. The alphanumeric name of a group.

Host. A computer system (such as the 8560) that is used to prepare and maintain programs that are tested and debugged on a workstation such as the 8540.

I-node. An entry in a TNIX data structure that describes:

- the physical location of a file on a fixed disk,
- the ownership and protection modes of a file,
- the last access date of a file, and
- the number of filenames associated with a file.

I/O Redirection. The ability of the TNIX shell to take the source and destination of a program's input and output from files other than standard input and standard output.

K. 1024 bytes (400 hexadecimal).

Kernel. The main memory-resident part of the TNIX operating system.

Keyshell. An interface to TNIX that enables you to enter commands by pressing function keys as well as typing the commands literally.

Library. A collection of object modules that usually contains commonly used subroutines.

Link. (1) A pointer from a directory to a file. The **ln** command allows you to create multiple pointers or links to a single file. (2) To merge object modules into a load module using the linker.

Linker. The system program that combines object modules into a single executable load module. The Tektronix Linker is described in the 8500 Modular MDL Series Assembler Core Users Manual for B Series Assemblers.

Logical Device. See special file.

Log In. To sign into the TNIX operating system.

Login Name. The name by which the system recognizes a particular user at login. A login name is typically 1 to 8 lowercase letters. The login name is also referred to as the username.

Major Device Number. An entry in a TNIX data structure that specifies the type of software device driver used to perform a data transfer operation. Each type of I/O device (i.e., a fixed disk drive or a terminal) uses the same software device driver, and therefore has the same major device number. Individual devices of each type are specified by the minor device number.

Make. Described in Section 6 of the 8560 Series System Users Manual, the **make** program can be used to perform the clerical tasks associated with updating and maintaining interrelated program modules. With moderately complex programs, this can be an almost indispensable time-saver and program bug-saver.

Man. A TNIX command that provides access to online information. For example, **man cd** displays information about the **cd** command.

Metacharacter. A character used to match specific patterns of characters. Metacharacters are used by the TNIX shell, **ed**, **sed**, **grep**, **awk**, and a number of other programs. For example, the TNIX shell substitutes any filenames that match the pattern of characters represented by the series of one or more metacharacters. Regular expressions (described later in this Glossary) are defined by a series of metacharacters.

Minor Device Number. An entry in a TNIX data structure that specifies the physical device to be accessed for a data transfer operation. For example, if there are two line printers connected to your system, TNIX would use the major device number to select the software device driver for the printer, and the minor device number (internally) to specify which printer to transfer data to.

Mode Bits. The 12 protection bits associated with a file. The first 9 bits specify read, write, and execute access for the owner, members of the owner's group, and remaining users. The last 3 bits specify that the program set the group/user ID on execution, and that the program remains in main memory for a minimum amount of time (use this mode for heavily used programs, such as **ls**). Use the **chmod** command to change a file's mode bits, and the shell's **umask** command to set the default file creation mode bits.

Mount. To associate a physical device (such as an 8503 Disk Expansion Unit) with a directory in the file tree. The data stored on the device is then accessible as subcomponents of the tree. See your 8560 Series System Manager's Guide for more information.

Multiplexed Special File. A special file that allows multiple communication paths between one or more executing processes.

Newline. The newline character is the ASCII LF (octal 12) character, and is used to separate lines in TNIX files. You can enter a newline character into a file by pressing the LINEFEED or RETURN key on your terminal.

OS/40. The operating system of the 8540 Integration Unit.

Parameter. A name or expression in a TNIX command line that supplies information to the command, command file, or program to be executed.

Parent. (1) The directory that contains a specific directory or file. For example, /usr/gandalf is the parent directory of /usr/gandalf/projects. (2) The originating process when an executing process creates another concurrently executing process with the TNIX system call **fork**. (The newly created process is called the child process.)

Password. The private sequence of characters that allows you to be identified to the system by your username. Each username may have an associated password, initially assigned by the system manager. You can change your password with the **passwd** command. Passwords are stored in an encrypted format in the /etc/password file.

Pathname. A sequence of directory names separated by slashes and ending in a filename, that defines a path to a file. For example, /usr/tektronix/myprog is a pathname for the file myprog in the directory tektronix. If /usr/tektronix is the current directory, myprog is a synonym for /usr/tektronix/myprog.

Permission. See execute permission, read permission, and write permission, in this Glossary.

Pipe. A communication channel used to transfer data between two executing programs. Pipes may be established from the shell using the "|" character.

Pipeline. A sequence of one or more commands separated by "|". The standard output of each command except the last is connected by a pipe to the standard input of the next command.

Process. An independently executing program with its own current directory, open files, user ID, group ID, and other process-specific information. Each process has its own memory for stacks, variables, and program information.

Process ID. A number in the range 1--30000 that uniquely identifies an executing process to TNIX.

Profile. (1) A file named .profile in your HOME directory that contains a list of commands executed by the TNIX shell when you first log into TNIX. (2) To obtain an execution trace of a program as an aid in debugging the program.

Prompt. The character that a program displays to tell you that it is waiting for input. The TNIX shell has three default prompt characters: "$", ">", and "#".

Queue. (1) To place files to be printed or processes to be executed into a waiting line. (2) The waiting line itself. Files are said to be "spooled" or "queued" when they are in a print queue.

Raw Device. A software device driver that transfers data without any processing by the system.

Read Permission. The ability to read from a file or list the contents of a directory. Permission is regulated by mode bits, and can be set with the **chmod** command.

Regular Expression. A series of one or more characters used to match specific patterns of characters. Regular expressions are used by the TNIX shell, **ed**, **sed**, **grep**, **awk**, and a number of other programs. Regular expression syntax varies from program to program. Be sure to consult the documentation for regular expression syntax for a specific program.

Regular File. Any file that is not a directory or a special file.

Root Directory. The directory (represented by "/") at the top (root) of the TNIX file tree. Except for the root directory itself, all files are descendants of the root directory.

Shell. A command interpreter that serves as an interface between you and the TNIX operating system. (See the TNIX Operating System section of the 8560 Series System Users Manual.)

Shell Procedure. A shell program. The Shell Programming section of the 8560 Series System Users Manual describes the shell programming language.

Shell Variable. Any command argument in a shell command line whose first character is a "$". Before executing the command specified in the command line, the TNIX shell replaces the specified shell variable with that variable's value.

Slave Computer. A microcomputer development system (such as the 8540) that is connected to a host computer (such as the 8560).

Special File. An entry in a directory, usually the /dev directory. Each special file is associated with a software device driver, and read, write, open, and close operations on a special file are actually performed by the associated software device. The TNIX operating system handles I/O to devices as if they were part of the file system. (See block special file, character special file, and multiplexed special file in this Glossary.)

Standard Error. A file descriptor (2) that is used by the TNIX shell to display error messages or other information that should not be intermingled with standard output. Normally sent to your terminal, the standard error output can be redirected to other files or devices.

Standard Input. A file descriptor (0) that is used by the TNIX shell to supply input to a program. Standard input is usually your terminal, but may be redefined with the <, <<, or | characters.

Standard Output. A file descriptor (1) that is used by the TNIX shell to display output from a program. Standard output is usually your system terminal, but may be redefined with the >, >>, or | characters.

String. A series of one or more ASCII characters.

Subdevice Number. See minor device number.

Subshell. A shell process created by a parent shell process. A subshell inherits the parent shell's environment, which includes the current directory, group ID, and variable settings, but may redefine its environment.

Superuser. Any user logged into the "root" account, usually to perform some system maintenance operation (such as file system backup, restore, or repair). See the 8560 Series System Manager's Guide for further information.

NOTE

Normal file protection does not apply to the superuser. Some system calls are also restricted for use only by the superuser.

System Call. An operating system service routine executed directly by the TNIX operating system. System calls are entered directly via software interrupts (traps) as rather than by the typical subroutine calling sequence.

System Manager. The person responsible for system operations such as system backups and restores, and software installation. The system manager usually logs into the "root" account in order to perform these operations. See the 8560 Series System Manager's Guide for further information.

Tekhex. Tektronix Hexadecimal Format: a format for representing the contents of a block of memory as an ASCII sequence of hexadecimal digits. An extended version of Tekhex supports 16-bit microprocessors and symbolic debug. (Refer to the 8540 or 8550 System Users Manual.)

TERM Mode. A mode of communication between an 8560 and an 8540 (or 8550), in which workstation commands can be entered just as if they were TNIX commands. The 8560 recognizes the workstation commands and passes them to the 8540 (or 8550). See the Communication with 8540s and 8550s section of 8560 Series System Users Manual for a detailed description of the TERM mode interface.

TNIX. The operating system of the 8560 Series Multi-User Software Development Unit.

Tree. A data structure. The TNIX filesystem structure is referred to as a file tree, because the data structure used by the TNIX filesystem is an inverted tree structure.

Unmount. To dissociate a physical device from a position in a file tree. The data stored on the device can no longer be accessed as subcomponents of the tree.

Unlink. To delete a pointer to a file from a directory. A file is destroyed when its last link is removed.

Upload. To transfer data from a workstation such as the 8540 or 8550 to a host, in this case, the 8560.

Username. The name by which the system recognizes a particular user at login. A username is typically 1 to 8 lowercase letters. The username is also referred to as the login name.

Wildcard Character. See metacharacter.

Word. A sequence of non-blank characters.

Working Directory. The directory that you are currently accessing. The working directory is used as a starting place when a full pathname (a name that begins with "/") is not given. The working directory is sometimes referred to as the current directory.

Write Permission. The ability to write to a file or to create and delete files in a directory. Permission is regulated by mode bits, and can be set with the **chmod** command.

Section 7

INDEX

3-byte integer to long integer conversion, 3-38
8540 command, 1-10
8540, running commands on, 1-10
8550 command, 1-10
8550, running commands on, 1-10
8550 flexible disks, 1-40
8560 flexible disks, 1-46, 1-64, 1-71, 1-151

A

a command in ed, 1-57
a.out file format, 5-2
abort subroutine, 3-7
abs subroutine, 3-8
absolute value, obtaining, 3-8, 3-21
access, checking file, 2-9
acos subroutine, 3-61
alarm system call, 2-10
allocation of memory, 2-11, 3-40
ampersand in ed, 1-59
appending one file to another, 1-161
ar file format, 5-6
arc:
  cosine function, 3-61
  sine function,3-61
  subroutine, 3-46
  tangent function, 3-61
arithmetic, subroutines for multiple precision integer, 3-39
asctime subroutine, 3-12
asin subroutine, 3-61
asm command, 1-11
assembly language, 2-1
assert subroutine, 3-9
assertions for program verification, 3-9
assigning buffering to a stream, 3-59
asterisk in ed, 1-59
atan subroutine, 3-61
atan2 subroutine, 3-61
atobobj command, 1-13
atof subroutine, 3-10
atoi subroutine, 3-10
atol subroutine, 3-10
aux, aux1, aux2 special files, 4-2

B

background processes, status of, 1-147

backslash in ed, 1-59
backup, file, 1-46, 1-64
backups, restoring, 1-64, 1-151
banproto file format, 5-7
baud rate, setting terminal, 1-175
Bessel functions, 3-37
binary I/O subroutines, 3-23
blocks, listing file size in,
blocks remaining, listing, 1-37
boolean expressions, evaluating, 1-190
boot block, making, 1-116
brackets in ed, 1-59
break system call, 2-11
breakpoint debugging, 2-41
brk system call, 2-11
buffered binary I/O subroutines, 3-23
buffering an input stream, 3-59
buffers, stream, closing and flushing, 3-19
bytes, swapping, 3-66

C

c command in ed, 1-57
cabs subroutine, 3-35
calloc subroutine, 3-40
caret in ed, 1-59
carriage returns on line printer, 1-167
cat command, 1-16
catching signals, 2-41
catenating files, 1-16
cbreak mode terminal I/O, 1-175
cd command, 1-17
ceil subroutine, 3-21
ceiling function, 3-21
changing:
  directories, 1-17, 2-12
  file ownership, 1-23, 2-14
  file and directory protection modes, 1-19 thru 1-22, 2-13, 2-52
  gpib device definitions, 1-122
  group id, 1-23
  kernel parameters, 1-29
  Keyshell scripts, 1-161
  memory allocation, 2-11
  passwords, 1-143
  user accounts, 1-129, 1-161
  user group assignments, 1-129
  users, 1-178
character classification, subroutines for, 3-27
character special devices, controlling, 2-27
characters:
  counting, 1-206
  pushing back to input stream, 3-69
  reading from a stream, 3-26

translating, 1-194
writing on a stream, 3-49 thru 3-53
chdir system call, 2-12
chgrp command, 1-23
checking the filesystem, 1-180
checklist file format, 5-9
child process, controlling and tracing, 2-41
chmod command, 1-19
chmod system call, 2-13
chown command, 1-23
chown system call, 2-14
chroot system call, 2-12
circle subroutine, 3-46
clearerr subroutine, 3-20
clearing error indication on a stream, 3-20
close system call, 2-15
closepl subroutine, 3-46
closing a file, 2-15
closing a stream, 3-19
cmp command, 1-24
ColorKey+ interface, 1-85, 1-161
comm command, 1-25
command index, 1-2
command page format, 1-8
command priority, changing, 1-140
commands, executing via system calls, 2-18
common lines, selecting and rejecting, 1-25
communication parameters of terminal, setting, 1-175
comparing files, 1-24, 1-38
comparing strings, 3-65
concatenating files, 1-16
concatenating strings, 3-65
cont subroutine, 3-46
control characters in a file, displaying, 1-141
converting:
between 3-byte and long integers, 3-38
numbers to ASCII digits, 3-16
strings to numbers, 3-10
cooked mode terminal I/O, 1-175
copying files and directories, 1-26
copying strings, 3-65
core dump, obtaining, 3-7
core file format, 5-10
cos subroutine, 3-61
cosh subroutine, 3-62
counting lines, words, and characters, 1-206
cp command, 1-26
CPU time, 1-193
creat system call, 2-16
creating:
directories, 1-117, 2-32
file systems, 1-119, 1-120
files, 2-16
gpib device definitions, 1-122

  groups, 1-128
  pipes, 2-38, 3-48
  processes, 2-22
cron command, 1-27
crontab syntax, 1-27
crypt subroutine, 3-11
ctime subroutine, 3-12
CTS flagging, setting for terminal,
cvt command, 1-29
cvt file format, 5-11

D

d command, in ed, 1-57
daemons, line printer, 1-97
data:
  base subroutines, 3-14
  sink file, 4-9
  structure size parameters, 1-31
date:
  adding to printouts, 1-145
  command, 1-35
  adding to printouts, 1-145
  printing, 1-35
  setting, 1-35, 2-50
dbminit subroutine, 3-14
ddate file format, 5-15
decimal notation, displaying files in, 1-141
defining gpib devices, 1-122
del50 command, 1-40
delaying execution,
delete subroutine, 3-14
deleting:
  accounts, 1-128
  directories, 1-128, 1-156, 2-53
  files, 1-156, 2-53
  file systems, 2-33
density, disk recording, 1-71
dependency lists, 1-110
DES encryption subroutines, 3-11
device definitions, gpib, 1-122
df command, 1-37
diff command, 1-38
differences between files, finding, 1-24, 1-25, 1-38
differential file comparison, 1-38
dir file format, 5-13
dir50 command, 1-40
direct special gpib file, 4-4
directories:
  changing working, 1-17, 2-12
  checking default protection mode of, 1-117
  copying, 1-26
  deleting, 2-53, 1-128, 1-156

displaying working, 1-150
listing contents of, 1-100
making, 2-32
moving, 1-138
protecting, 1-19 thru 1-22
removing, 2-53, 1-128, 1-156
renaming, 1-138
size of, 1-45, 1-100
unlinking, 1-156, 2-53
directory, changing root, 2-12
directory, of 8550 disk, 1-40
discarding output, 4-9
disk space used, 1-37
disk usage statistics, 1-37
disks, 8550 flexible, 1-40
disks, 8560 flexible, 1-46, 1-64, 1-71, 1-151
disks, formatting, 1-71
displaying:
control characters in a file, 1-141
files, 1-16, 1-133
kernel parameters, 1-29
man pages, 1-113
process status, 1-147
the name of your terminal, 1-199
See also printing
dist file format, 5-14
distance, finding Euclidean, 3-35
documentation, online, 1-80, 1-113
dollar sign in ed, 1-59
double-density formatting, 1-71
downloading files, 1-131, 1-200
downloading, unformatted, 1-200
dsc50 command, 1-40
DTR flagging, setting for terminal,
du command, 1-45
dump command, 1-46
dump file format, 5-15
dump -- octal, decimal, and hexadecimal, 1-141
dumpdir command, 1-53
dumps, core, 3-7
dumps, incremental, 1-49
dup system call, 2-17
dup2 system call, 2-17
duplicating file descriptors, 2-17

E

E command in ed, 1-58
e command in ed, 1-57
echo command, 1-54
echo, setting terminal, 1-175
echoing strings, 1-54
ecvt subroutine, 3-16

ed command, 1-55
ed command quick reference, 1-57
edata subroutine, 3-17
egrep command, 1-75
encrypt subroutine, 3-11
encryption keys, command that makes, 1-112
encryption, subroutines for DES, 3-11
end of file associated with a stream, 3-20
end subroutine, 3-17
endgrent subroutine, 3-28
endpwent subroutine, 3-33
environ file format, 5-18
environ system call, 2-18
environment list, searching, 3-27
erase characters, setting for terminal, 1-175, 1-197
erase subroutine, 3-46
erasing. See deleting
error messages for system calls, 2-2
error messages, printing, 3-44
error, detecting on stream, 3-20
error, resetting indication on stream, 3-20
etext subroutine, 3-17
Euclidean distance, finding, 3-35
evaluating expressions, 1-61, 1-189
exchanging bytes, 3-66
exec system call, 2-18
exece system call, 2-18
execl system call, 2-18
execle system call, 2-18
execute permission, 1-19
executing commands via system calls, 2-18 thru 2-19
execution time, profiling, 2-39, 3-42
execv system call, 2-18
execve system call, 2-18
exit status variable, setting, 1-196
exit system call, 2-21
exp subroutine, 3-18
exponent, returning, 3-24
exponential function, 3-18
expr command, 1-61
expressions, evaluation of, 1-61, 1-189

F

fabs subroutine, 3-21
false command, 1-196
fbr command, 1-64
fbr file format, 5-19
fclose subroutine, 3-19
fcvt subroutine, 3-16
fd special file, 4-3
fdopen subroutine, 3-22
feof subroutine, 3-20

ferror, 3-20
fetch subroutine, 3-14
fflush subroutine, 3-19
fgetc subroutine, 3-26
fgets subroutine, 3-34
fgrep command, 1-75
file:
  descriptor, associated with a stream, 3-20
  descriptors, duplicating, 2-17
  formats, Section 5
  names, making unique, 3-41
  ownership, changing, 1-23, 2-14
  protection, 1-19, 2-13, 2-52
  status, 2-46
file system:
  backing up, 1-46, 1-64
  checking, 1-180
  making, 1-119, 1-120
  mounting and unmounting, 1-136, 2-33
  removing, 2-33
fileno, 3-20
files:
  backing up, 1-46, 1-64
  changing ownership, 1-23, 2-14
  closing, 2-15
  comparing, 1-24, 1-38
  copying, 1-26
  counting lines, words, and characters in, 1-206
  creating, 2-16
  displaying, 1-16, 1-133
  displaying the last part of, 1-187
  finding, 1-68
  finding differences between, 1-24, 1-38
  formatting for the line printer, 1-145
  linking, 1-93, 2-29
  listing, 1-100
  making unique names for, 3-41
  making special, 2-32
  moving, 1-138
  moving read/write pointer in, 2-31
  opening, 2-16, 2-35
  printing, 1-99, 1-145
  protecting, 1-19, 2-13, 2-52
  reading from, 2-42
  removing, 2-53, 1-156
  renaming, 1-138
  restoring from backups, 1-64, 1-151
  searching for a pattern, 1-59, 1-75, 1-134
  size of, 1-45, 1-100
  sorting, 1-169
  special, Section 4
  standard input, output, and error, 3-64
  status of, 2-46
  unique lines in, 2-202

unlinking, 1-156, 2-53
updating modification time of, 2-54
writing in, 2-56
See also streams
filesystem. See file system
filsys file format, 5-21
find command, 1-68
finding files, 1-68
firstkey subroutine, 3-14
fixed disk formatting, 1-71
flagging, setting for terminal, 1-175
flblk file format, 5-21
flexible disk special file, 4-3
flexible disks:
8550, 1-40
8560, 1-46, 1-64, 1-71, 1-151
formatting, 1-71
making backups with, 1-40, 1-46, 1-64
floor function, 3-21
floor subroutine, 3-21
floppy disks. See flexible disks
flushing a stream, 3-19
fopen subroutine, 3-22
fork system call, 2-22
format command, 1-71
format conversion and printing routines, 3-49
formatting disks, 1-71
fprint subroutine, 3-49
fprintf subroutine, 3-49
fputc subroutine, 3-52
fputs subroutine, 3-53
fractional part of value, returning, 3-24
fread subroutine, 3-23
free blocks, listing number of, 1-37
free subroutine, 3-40
freopen subroutine, 3-22
frexp subroutine, 3-24
fscanf subroutine, 3-56
fseek subroutine, 3-25
fstat system call, 2-46
ftell subroutine, 3-25
ftime system call, 2-50
function calls, counting, 3-42
fwrite subroutine, 3-23

G

gcd subroutine, 3-39
gcvt subroutine, 3-16
general terminal interface, 4-10
getc subroutine, 3-26
getchar subroutine, 3-27
getegid system call, 2-23

getenv subroutine, 3-27
geteuid system call, 2-25
getgid system call, 2-23
getgrent subroutine, 3-28
getgrgid subroutine, 3-28
getgrnam subroutine, 3-28
getlogin subroutine, 3-30
getpass subroutine, 3-31
getpid system call, 2-23
getpw subroutine, 3-32
getpwent subroutine, 3-33
getpwnam subroutine, 3-33
getpwuid subroutine, 3-33
gets subroutine, 3-34
getting a process identification, 2-23
getty command, 1-73
getuid system call, 2-24
getw subroutine, 3-26
glossary, Section 6
gmtime subroutine, 3-12
gotos, non-local, 3-60
gpib:
  file format, 5-24
  device definitions, making, modifying, and printing, 1-122
  status information, 1-172
  interface, initializing, 1-87
  special files, 4-4
  tape drive characteristics, 1-158
graphics subroutines, 3-47
grep command, 1-75
group:
  entry in passwd file, reading, 3-28
  file entry, reading, 3-28
  file format, 5-25
  file, closing, 3-28
  file, rewinding, 3-28
  ID of a process, 2-24
  ID, changing, 1-23
  ID, reading, 3-28
  logging in to a new, 1-139
  making a new, 1-128
  membership, changing a user's, 1-23
  name, reading, 3-28
gtty system call, 2-26

H

hangups, making commands immune to, 1-140
hd special files, 4-5
headers, putting on files, 1-145
help command, 1-79
hexadecimal notation, displaying files in, 1-141
hexadecimal uploading and downloading, 1-131

hsi and hsix high-speed interface files, 4-7
hyperbolic functions, subroutines for calculating, 3-62
hypot subroutine, 3-35

I

i command in ed, 1-57
i-numbers, listing, 1-100
ignoring signals, 2-44
incremental dumps, 1-49
index command, 1-80
index subroutine, 3-65
indir system call, 2-25
init command, 1-81
initializing:
  system processes, 1-81
  terminals, 1-197
  the gpib interface, 1-87
ino file format, 5-21
input conversion subroutines, 3-56
install command, 1-83
installing software, 1-83
instr special gpib file, 4-4
integer arithmetic, subroutines for multiple precision, 3-39
internal disk formatting, 1-71
internal disk special files, 4-5, 4-6
invoking a shell, 1-162
ioctl system call, 2-26
IOT instruction (trap, fault), generating, 3-7
isalnum subroutine, 3-36
isalpha subroutine, 3-36
isascii subroutine, 3-36
isatty subroutine, 3-68
iscntrl subroutine, 3-36
isdigit subroutine, 3-36
islower subroutine, 3-36
isprint subroutine, 3-36
ispunct subroutine, 3-36
isspace subroutine, 3-36
isupper subroutine, 3-36
itom subroutine, 3-39
IU, setting port number, 1-175

J

j command in ed, 1-57
j0/j1/jn subroutines, 3-37
jump subroutines, 3-60

K

k command in ed, 1-57
kernel parameters, examining and changing, 1-29
keys, making encryption, 1-112
Keyshell interface, 1-85, 1-161
keyword index to online manuals, 1-80
kill:
  characters, setting for terminal, 1-175, 1-197
  command, 1-84
  system call, 2-28
killing processes, 1-83, 1-165, 2-21, 3-7
kmem special file, 4-8
ksh command, 1-85, 1-161

L

l command in ed, 1-57
l3tol subroutine, 3-38
label, 3-46
ldexp subroutine, 3-24
ldgpib command, 1-87
lf command, 1-100
libgen command, 1-89
libraries, 3-1
library generator, 1-89
line numbers, printing with, 1-145
line printer:
  changing output characteristics, 1-167
  daemons controlling the, 1-97
  sending files to, 1-99
  special files, 4-2
linemod subroutine, 3-46
lines, counting, 1-206
lines, finding unique, 1-202
link command, 1-91
link system call, 2-29
linking files, 1-93, 2-29
linking object modules, 1-91
listing:
  directory contents, 1-100
  files in a directory, 1-100
  free space on a disk, 1-37
  symbols in object modules, 1-104
  See also displaying, printing
ll command, 1-100
ln command, 1-93
ln50 command, 1-40
load module, creating, 1-91
localtime subroutine, 3-12
locations, program, finding, 3-17
lock system call, 2-30
locking processes in memory, 2-30

log subroutine, 3-18
log10 subroutine, 3-18
logarithm, finding, 3-18
logging in, 1-95, 1-178
logging in to a new group, 1-139
login command, 1-95
login name, reading, 3-30
long integer to 3-byte conversion, 3-38
longjmp subroutine, 3-60
lost+found directory, 1-182
lp1d and lp2d line printer daemons, 1-97
lp1r command, 1-99
lp2r command, 1-99
lpd command, 1-97
lpr command, 1-99
lr command, 1-100
ls command, 1-100
lseek system call, 2-31
lstr command, 1-104
ltol3 subroutine, 3-38
lx command, 1-100

M

m command in ed, 1-57
madd subroutine, 3-39
magnetic cartridge tape block capacities, 1-52
magnetic reel-to-reel tape block capacities 1-51
magtape special, pib file, 4-4
mail command, 1-106
mail, sending and receiving, 1-106
make command, 1-110
makefile, 1-110
makekey command, 1-112
making:
  directories, 1-117, 2-32
  file systems, 1-119, 1-120
  gpib device definitions, 1-122
  groups, 1-128
  new accounts, 1-128, 1-161
  special files, 2-32
malloc subroutine, 3-40
man command, 1-113
man pages, online access to, 1-80, 1-113
mantissa of a number, returning, 3-24
manuals, online, 1-80, 1-113
mapping virtual to physical addresses, 2-37
mdiv subroutine, 3-39
mem special file, 4-8
memory:
  allocation, 2-11, 3-40
  image special file, 4-8
  locking processes in, 2-30

physical, 3-36
status of a process, 1-147
merging files, 1-169
mesg command, 1-114
message permission, changing, 1-114
messages:
sending to other users, 1-209
sending to all users, 1-205
See also mail
metacharacters, shell, 1-164
min subroutine, 3-39
mk50 command, 1-40
mkboot command, 1-116
mkdir command, 1-117
mkfs command, 1-119, 1-20
mkgpib command, 1-122
mkgroup command, 1-127
mknod system call, 2-32
mktemp subroutine, 3-41
mkuser command, 1-128
mload command, 1-130
modes:
changing default file protection, 2-52
changing file and directory protection, 1-19, 2-13, 2-52
checking default directory protection, 1-117
listing file, 1-100
modf subroutine, 3-24
modifying. See changing
monitor special, gpib file, 4-4
monitor subroutine, 3-42
more command, 1-132
mount command, 1-136
mounting and unmounting file systems, 1-136
mout subroutine, 3-39
move subroutine, 3-46
moving file read/write pointers, 2-31
moving files and directories, 1-138
msub subroutine, 3-39
mtab file format, 5-26
mult subroutine, 3-39
multiple precision integer arithmetic, subroutines for, 3-39
mv command, 1-138

N

name list, reading system, 3-43
name, reading login, 3-30
newgrp command, 1-139
new group, logging in to a, 1-139
newlines on line printer, 1-167
new users, making accounts for, 1-128, 1-161
nextkey subroutine, 3-14
nice command, 1-140

nice system call, 2-34
nlist subroutine, 3-43
nohup command, 1-140
non-local, gotos, 3-60
notation conventions, standard TNIX commands, 1-9
null special file, 4-9

O

object module format conversion, 1-13
object module librarian, 1-89
object modules, linking, 1-91
object modules, listing symbols in, 1-104
octal dump, 1-141
octal notation, displaying files in, 1-141
octal permission modes, 1-20
od command, 1-141
open system call, 2-35
opening a file, 2-16, 2-35
openpl, 3-46
ownership of files, changing, 1-23, 2-14

P

P command in ed, 1-57
p command in ed, 1-57
packet driver subroutines, 3-45
page numbers, printing with, 1-145
pager, file, 1-133
pagination, 1-133, 1-145
parity, setting terminal, 1-175
passwd command, 1-143
password:
  encryption, 3-11
  changing, 1-143
  determining from user ID, 3-32
  reading, 3-31, 3-33
  use with login, 1-95
password file:
  format of, 5-27
  rewinding, 3-33
  searching, 3-33
  subroutines for reading, 3-33
pause, 2-36
pclose subroutine, 3-48
period in ed, 1-59
permission to receive messages, 1-114
permissions, changing file and directory, 1-19, 2-13, 2-52
permissions, listing file access, 1-100
perror subroutine, 3-44. See also error messages from system calls
phys system call, 2-37
physical addresses, accessing,2-37

pipe fitting, 1-189
pipe system call, 2-38
pkclose subroutine, 3-45, 3-48
pkfail subroutine, 3-45
pkopen subroutine, 3-45
pkread subroutine, 3-45
pkwrite subroutine, 3-45
plot file format, 5-28
plot subroutine, 3-46
plotting subroutines, 3-46
pmuboot file, 1-116
point subroutine, 3-46
pointer, moving in a file, 2-31
popen subroutine, 3-48
pow subroutine, 3-18, 3-39
powers, finding, 3-18, 3-39
pr command, 1-145
preventing display of messages, 1-114
print and format conversion subroutines, 3-49 thru 3-53
printer. See line printer
printf subroutine, 3-49
printing:
  error messages, 3-44
  files, 1-99, 1-145
  gpib device definitions, 1-122
  strings, 3-49, 3-53
  with headers, line numbers and page numbers, 1-145
  your working directory, 1-150
  See also displaying
priority of a command, changing, 1-140
priority, displaying process, 1-147
process, awaiting completion of, 1-147, 1-204
processes:
  communicating between, 2-38
  creating, 2-22
  displaying and interpreting status of, 1-147
  group ID of, 1-147, 2-24, 2-43
  initializing system, 1-81
  killing, 1-83, 1-165, 2-21, 3-7
  locking in memory, 2-30
  priority of, 1-147
  pausing in, 2-36
  sending signals to, 2-28
  size of, 1-147
  state of, 1-147
  suspending execution of, 3-63
  timing, 2-51
  tracing, 2-41
  user ID of, 1-147, 2-24, 2-43
  waiting for termination of, 1-204, 2-55
profil system call, 2-39
profile (.profile) file, 1-129, 1-161, 1-197
profiling program execution, 2-39, 3-42
program locations (addresses), finding, 3-17

protecting:
  directories, 1-19, 2-13
  files, 1-19, 2-13
  See also modes
protocol, files for HSI, 4-7
ps command, 1-147
ptrace system call, 2-40
pushing characters back onto the input stream, 3-69
putc subroutine, 3-52
putchar subroutine, 3-52
puts, 3-53
putw subroutine, 3-52
pwd command, 1-150

Q

qsort (quicker sort) subroutine, 3-54

R

r command in ed, 1-57
rand subroutine, 3-55
random number generator, 3-55
raw disk I/O, 4-3, 4-5, 4-6
raw mode terminal I/O, 1-175
rd50 command, 1-40
read permission, 1-19
read system call, 2-42
read/write pointers, moving in a file, 2-31
reading 8550 disks, 1-40
reading from a file, 2-42
reading mail, 1-106
real user ID,, getting, 2-25
realloc subroutine, 3-40
reallocating memory, 3-40
regular expressions, searching for, 1-59, 1-75
removing:
  accounts, 1-128
  directories, 1-128, 1-156, 2-53
  files, 1-156, 2-53
  file systems, 2-33
repeated lines in file, reporting, 1-202
repositioning a stream, 3-25
restor command, 1-151
restoring backups, 1-64, 1-151
rewind subroutine, 3-25
rfd0 special file, 4-3
rindex subroutine, 3-65
rm command, 1-156
rmdir command, 1-156
root device parameters, 1-32
root directory, changing, 2-12

rpow subroutine, 3-39

S

s command in ed, 1-57
sbrk system call, 2-11
scanf subroutine, 3-56
scheduling priority of a command, changing,
scripts, cvt, 1-29, 5-11
scripts, Keyshell, 1-161
search and regular expression quick-reference, 1-59
searching:
  files, 1-59, 1-75
  for files, 1-68
  the environment list, 3-27
sending and receiving mail, 1-106
setbuf subroutine, 3-59
setgid system call, 2-43
setgrent subroutine, 3-28
setjmp subroutine, 3-60
setkey subroutine, 3-11
setpwent subroutine, 3-33
settape command, 1-158
setting:
  communication parameters of a terminal, 1-175, 2-26
  group ID of a process, 2-43
  terminal options, 1-175, 2-26
  the date, 1-35
  the time, 1-35, 2-48
  user ID, 2-43
setuid system call, 2-43
setuser command, 1-161
sh command, 1-162
shell:
  commands, issuing from C programs, 3-67
  metacharacters, 1-164
  programming commands, 1-162
  invoking a, 1-162
  setting the exit status of, 1-196
shutdown command, 1-165
signal scheduling, 2-10
signal system call, 2-44
signals, catching and ignoring, 2-44
signals, list of, 2-44
sin subroutine, 3-61
sine function, 3-61
single-density formatting, 1-71
sinh subroutine, 3-62
size of files, displaying, 1-100
sleep command, 1-166
sleep subroutine, 3-63
slp command, 1-167
sort command, 1-169

sorting and merging files, 1-169
sorting, subroutine for, 3-54
space on disk, listing, 1-37
space subroutine, 2-32, 3-46
special characters in ed, 1-59
special files, Section 4
spooler, line printer, 1-97, 1-199
sprintf subroutine, 3-49
sqrt subroutine, 3-18
square root, finding, 3-18
srand subroutine, 3-55
sscanf subroutine, 3-56
standard:
  input/output package, 3-64
  subroutines, Section 3
  TNIX commands, Section 1
  TNIX commands, classified list of, 1-2
starting system processes, 1-81
stat system call, 2-46
statgpib command, 1-172
status:
  of a file, examining, 2-46
  of a gpib device, 1-172
  of processes, displaying, 1-147
stdin, stdout, stderr, 3-64
stdio package, 3-64
stime system call, 2-48
stopping processes, 1-83, 1-165, 2-21, 2-35, 3-7
stopping the system, 1-165
storage allocation and compaction, 2-11, 3-40
store subroutine, 3-14
strcat subroutine, 3-65
strcmp subroutine, 3-65
strcpy subroutine, 3-65
stream:
  closing and flushing, 3-19, 3-23, 3-48
  detecting EOF on, 3-20
  detecting error on, 3-20
  file descriptor associated with, 3-20
  opening, 3-23, 3-48
  printing on, 3-49 thru 3-53
  pushing characters back onto, 3-69
  reading a string from, 3-34, 3-56
  repositioning, 3-25
  resetting error indication on, 3-20
  standard, 3-64
  substituting for, 3-23
  See also files
string-handling subroutines, 3-65
strings:
  comparing, concatenating, and copying, 3-65
  converting to numbers, 3-10
  finding position in an array, 3-65
  reading from a stream, 3-34, 3-56

  writing, 1-54, 3-49, 3-53
strlen subroutine, 3-65
strncat subroutine, 3-65
strncmp subroutine, 3-65
strncpy subroutine, 3-65
stty command, 1-175
stty system call, 2-26
su command, 1-178
subroutine index, 3-3
subroutines, standard, Section 3
subshells, invoking, 1-162
substituting characters, 1-194
substituting user ids, 1-178
superblock, updating, 1-80, 1-203, 2-49
suspending execution of a process, 1-166, 3-63
swab subroutine, 3-66
swap parameters, 1-33
swapping bytes, 3-66
symbols in object files, listing, 1-104
sync command, 1-179
sync system call, 2-49
syschk command, 1-180
system:
  call error messages, 2-2
  calls, Section 2
  calls, indirect, 2-25
  date and time, setting and reading, 1-35, 2-50
  name list, reading, 3-43
  processes, starting, 1-81
  subroutine, 3-67

T

t command in ed, 1-57
tabs on line printer, 1-167
tail command, 1-187
tan subroutine, 3-61
tangent function, 3-61
tanh subroutine, 3-62
tape, gpib drive characteristics, 1-158
tape, making backup copies on, 1-46
tee command, 1-189
TEKHEX file transfers, 1-131
tell system call, 2-31
termcap file, 1-197
termcap file format, 5-30
terminal:
  communication parameters, setting, 1-175, 2-26
  interface special file, 4-10
  name, obtaining, 1-199
  options, displaying,
  options, setting, 1-175, 1-197
  parameters, displaying, 1-23

displaying the name of your, 1-199
See also tty
terminals:
controlling, 1-175, 1-197, 2-26
file descriptors and, 3-68
initializing, 1-175, 1-197, 2-26
terminate signals, making commands immune to, 1-140
terminate, waiting for a process to, 1-204, 2-55
terminating processes, 2-21, 3-7
test command, 1-190
text editor ed, 1-55
time:
command, 1-192
executing a file at a specific, 1-27
printing, 1-35
profiling execution, 2-11, 3-43
setting date and, 1-35, 2-50
system call, 2-50
times system call, 2-51
timezone subroutine, 3-12
timing a process, 2-51
timing a program, 1-192, 2-39
touch command, 1-193
tr command, 1-194
transfers to/from 8550 files, 1-40 thru 1-45
transfers, TEKHEX file,
translating characters, 1-194
trigonmetric functions, subroutines for calculating, 3-61
true command, 1-196
tset command, 1-197
tty:
characteristics, reading, 1-77
command, 1-199
special file, 4-9
See also terminal
ttyname subroutine, 3-68
ttys file format, 5-40
ttyslot subroutine, 3-68

U

u command in ed, 1-57
uload command, 1-200
umask system call, 2-52
umount command, 1-136
umount system call, 2-33
unmounting file systems, 1-136, 2-33
unformatted up/down load,
ungetc subroutine, 3-69
uniq command, 1-202
unique file names, making, 3-41
unlink system call, 2-53
unlinking (removing) directory entries, 2-53

update command, 1-203
updating files automatically, 1-110
updating modification time of a file, 1-193
updating the superblock, 1-179, 1-203, 2-49
uploading files, 1-131, 1-200
uploading, unformatted, 1-200
user ID of a process, obtaining the, 2-24
user ID, determining password from, 3-32
users, listing, 1-207
user time, 1-192
utime system call, 2-54
utmp file format, 5-41

W

w command, in ed, 1-57
wait command, 1-204
wait system call, 2-55
wall command, 1-205
wc command, 1-206
who command, 1-207
who is on the system, listing, 1-207
words:
  counting, 1-206
  reading from a stream, 3-26
  writing on a stream, 3-49, 3-52
working directory, changing, 1-17
working directory, displaying, 1-150
wr50 command, 1-40
write:
  command, 1-209
  permission, 1-19
  system call, 2-56
writing:
  8550 disks, 1-140
  in a file, 2-56
  to all users, 1-205
  to other users, 1-209
wtmp file format, 5-41

X

XON/XOFF flagging, setting for terminal, 1-175

Y

y0/y1/yn subroutines, 3-37

MANUAL CHANGE INFORMATION

At Tektronix, we continually strive to keep up with latest electronic developments by adding circuit and component improvements to our instruments as soon as they are developed and tested.

Sometimes, due to printing and shipping requirements, we can't get these changes immediately into printed manuals. Hence, your manual may contain new change information on following pages.

A single change may affect several sections. Since the change information sheets are carried in the manual until all changes are permanently entered, some duplication may occur. If no such change pages appear following this page, your manual is correct as printed.

83927